LEO SATELLITE VISIBILITY CLASH RESOLUTION USING GREEDY SEARCH, TABU SEARCH, SIMULATED ANNEALING AND GENETIC ALGORITHM

A Thesis Submitted

In Partial Fulfilment of the Requirements

For the Degree of

MASTER OF TECHNOLOGY

By

SAGAR RAMESH KAPSE

to the

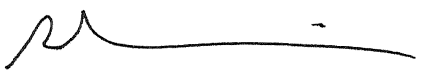
DEPARTMENT OF INDUSTRIAL AND MANAGEMENT ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL 2001

CERTIFICATE

This is to certify that the work contained in the thesis entitled " *LEO Satellite Visibility Clash Resolution using Greedy Search, Tabu Search, Simulated Annealing and Genetic Algorithms*" has been carried out by Sagar Ramesh Kapse (Roll No. 9911408) under my supervision and that this work has not been submitted elsewhere for any degree.



T. P. Bagchi
Professor,
Dept. of Industrial and Management Engineering,
Indian Institute of Technology,
Kanpur – 208016.

18 April, 2001

ABSTRACT

Satellite operations scheduling, like many other real life-scheduling undertakings, is a complex problem. Although NP-complete, it is more challenging than, say, job shop or open shop, due to the dynamic nature of the constraints involved. It is difficult to derive here the objective function(s). Many qualitative decisions are also involved. Therefore, this problem cannot be solved by conventional optimisation methods and the availability of many textbook models appears to be of little direct utility in satellite operations scheduling.

In this study, we consider the problem of optimally resolving satellite visibility clashes over a ground station. Such opportunities develop about 250 to 270 times each week in ISRO's LEO satellites network consisting of 8-10 ground stations, all individually tracking and communicating with the satellites. We propose a non-linear objective function for scheduling support of TTC type of operations. It also covers the issue of dynamic priority. Then three meta-heuristic approaches, viz. Greedy (local) Search, Tabu Search and Simulated Annealing are used to tackle the clash resolution problem. Each of these methods is able to find solutions for clashing satellite visibilities that are optimal or near to optimal. To use local search and yet avoiding getting trapped in a local optima, we used a new method of generating diversely dispersed initial solutions, one in each disjoint search region. This enabled reaching near-to-global optima in all runs. The study concluded with the comparison of the three above approaches and the GA approach used earlier to solve the clash resolution problem. Sample program codes in JAVA are attached.

ACKNOWLEDGEMENT

I would like to express my sincere thanks to Dr. T. P. Bagchi for his invaluable guidance and encouragement during the course of this thesis work. I am heartily thankful to him for the confidence he showed in me. I would also like to thank Dr. Sanjeev Swami and Dr. A.K. Mittal who introduced me the fundamentals of optimization and scheduling.

This work could not reach this stage without the direct consultation and advisory support of ISRO staff at ISTRAC Bangalore. The generous help given by Mr. P. Soma, Group Director MOHA (ISTRAC), Mr. Venkateswarlu, Mr. J. D. Rao, Mrs. Santhalakshmi and Mrs. Padmarani was critical in the success of this project. I thank them all.

I would specially thank to Sanjay, Garima and Baba who always guided and helped me at critical times in this study.

I would also like to cherish the nice company of IME lab "family" members Punit, Rama, Vishi, Batra, Rajku L, Vivek, Makarand, Shobhit, Shirish and above all the PM club members without which the life in IIT would have been monotonous. I would not forget the moments spent with first year M.Tech students. I am also grateful to Sanjay Kumar (Ph.D. scholar) for providing me various kinds of help in the lab during the course of my thesis work.

Last but not the least, I extend my thanks to all Professors, my colleagues and the whole IME family who made my stay in IIT enjoyable.

-Sagar

TABLE OF CONTENTS

LIST OF FIGURES/TABLES

FIGURES

TABLES

Chapter 1
The Challenge of Scheduling Satellite Operations

1.1 Introduction to Satellites and their Motion

A satellite is an object which revolves around another object. For example, the moon is a satellite of the earth and the earth is a satellite of the sun. Man-made or artificial satellites are those which orbit round the Earth in their fixed orbits. The shape of this orbit may be elliptical or close to circular.

India is currently having two series of satellites

- Indian National Satellite (INSAT) Series
- Indian Remote Sensing (IRS) Series

Satellites of the INSAT series including the latest launch of GSAT1 by GSLV1 are geo-stationary, moving at a height of around 36000 km, and therefore having a time-period of 24 hrs. These satellites are used for communication, broadcasting and meteorological purposes.

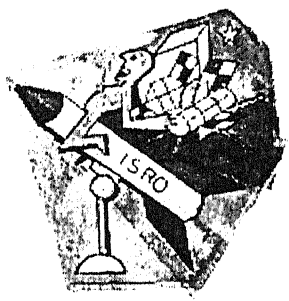
Satellites of IRS series are low earth orbit (LEO) satellites moving at a height of 700 to 900 km. Their time period is around 100 minutes. IRS satellites are used for a large number of remote sensing applications in areas such as agriculture and crops management, irrigation planning, forest and bio-resources management, ocean study, rural development, urban planning, etc. The present constellation of satellites of IRS series consists of 1A, 1B, 1C, 1D, P2, P3, P4, and SROSS.

Most Indian satellites are fabricated at ISRO Satellite Centre (ISAC), Bangalore while the launch vehicles are fabricated at Tiruvananthpuram. As stated above, Low Earth Orbit (LEO) satellite orbits at 700-900 kilometres above the earth, resulting in very low latencies. While this seems to be a simple solution, it is not that simple to implement LEO technology. As one brings the satellite closer to earth, the number of satellites needed to cover all areas on the earth increase and high-speed communication handoffs between satellites become complex. A LEO satellite may be visible for half an hour before it needs to hand-off the signal to the next incoming

satellite. This means that the antenna at the customer's premises has to be sufficiently intelligent to detect and switch from the passing satellites.'

1.1.1 Scheduling the Launch [13]

The launching of Indian satellites into required orbit is done from Shriharikota. Launching hundreds of multimillion-dollar satellites is a challenge. Teledesic had initial plans of a constellation of 840 satellites but that has now dropped to 288. It plans to launch all its satellites within a two-year time frame; Celestri has similar plans. A staggering 1,700 satellites will be launched in the next decade--more than 10 times the 150 commercial satellites now in orbit. This creates big capacity problems for launching these satellites. The existing services just do not have the capacity. In 1998 alone, we saw more than 60 launch carrying more than 100 satellites into orbit. To exacerbate things further, the launch reliabilities haven't improved since the dawn of space age and 1 in 10 launches fail. This makes the insurance premium on a satellite skyrocket to the point where it begins to rival the cost of the satellite.



The main purpose of IRS series of satellites is nation's resource monitoring and its management. Also, satellites generate revenues by taking images of various locations on the globe as requested by global customers.

Although the path of a satellite is a fixed orbit, by virtue of spin motion of earth, its ground trace is spiral shaped. Also, the spiral traversed each day is not exactly the same as that of previous day. The spiral path covered on a particular day is repeated after a fixed period of time, which may be different for different satellites. This period of time is called 'repeat cycle' of the spacecraft. (IRS-P4 → 2 days, IRS-1C → 24 days, etc.) In other words, repeat cycle of a satellite is a set of different paths the satellite can follow.

As explained above, a satellite covers and serves different points of globe at different points of its motion. Therefore, it is very important to monitor the health parameters of various sub-systems of the satellite continuously and also keep track of its orbit. This is achieved by means of special "chain" or ground station configurations set up at geographically distributed locations that can establish radio-

link with the spacecraft and thus can communicate (send and receive signals) with it. These configurations are commonly called *ground-stations.*

Ground-stations are exclusively of two types-

- Telemetry, Tracking and Commanding (TTC) stations, and
- Payload (PL) stations.

A set of two or more stations is called *ground network.*

1.1.2 Advancements in Micro Satellites: [15]

Micro satellites provide a number of unique advantages not found in larger satellites, but also suffer from a number of constraints. The most significant advantage is the cost of launching which is around $6 000 per kg compared to $40 000 per kg for larger satellites. Given the cost advantage and the miniaturisation of electronics and mechanics to build in the required functionality, the constraints amount to power availability and opportunity for launching.

The major limitations on micro satellites arises because of following constraints

1. Orbits and launchers
2. Power
3. Construction
4. Architectures

1.2 Operations on Satellites

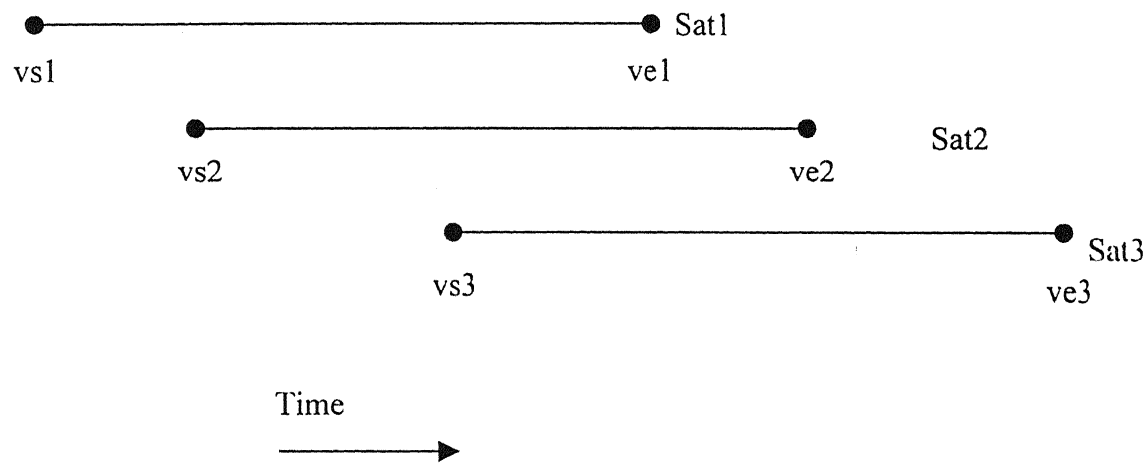
Satellite operations may be classified as follows:

- *TTC operations:* These operations are required for the health monitoring of the satellite subsystems.
- *Payload operations:* Payload operations are used in imaging the ground from LEO satellites.
- *Command Operations:* These are of very small duration and are operations performed to send signals to the space craft for resetting its various devices, switching ON/OFF the cameras, commanding the timers, etc.

All TTC stations are equipped with almost identical systems of telemetry, reception, tracking and commanding. The TTC network also provides dedicated services for payload operations and spacecraft health keeping throughout the mission life. Payload stations receive data from spacecrafts, as scheduled.

1.3 What is a "clash"?

In MSS software documentation [17] the definition of a visibility clash given by ISTRAC is "Over a ground station if two or more spacecrafts pass simultaneously or one after the other such that the time difference between first spacecraft LOS and second spacecraft AOS is less than the station reconfiguration time, then the two spacecrafts are said to have a **visibility clash**".

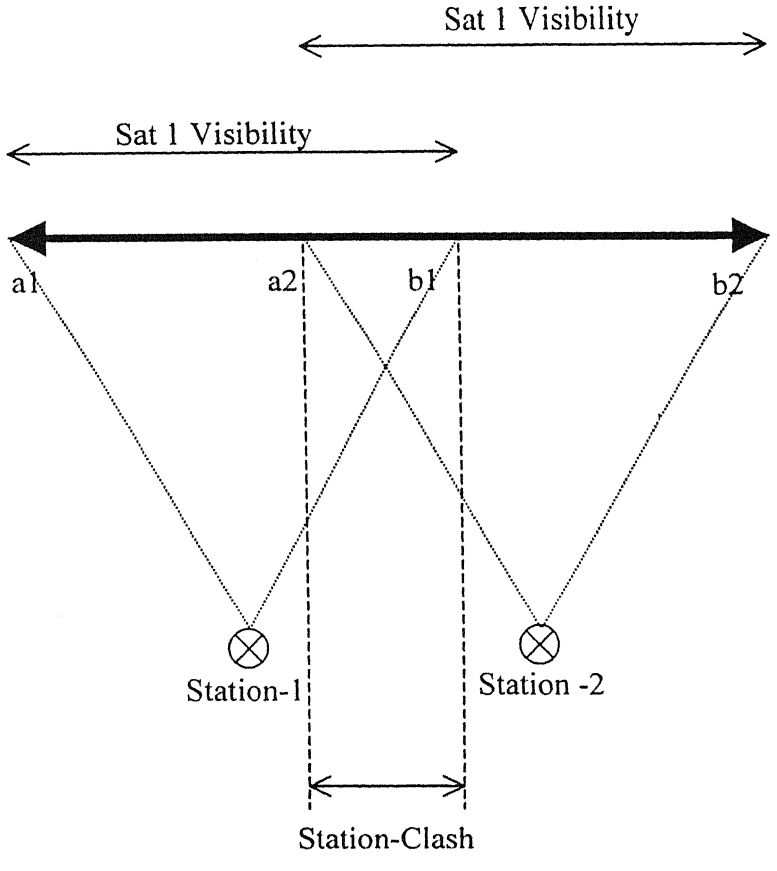


vsi visibility start time of i^{th} satellite.

vei = visibility end time of i^{th} satellite

Fig 1.1 Visibility Clash

Also when a single satellite is visible from two or more ground stations, its important to decide which station will support that particular satellite. This situation is called a **station clash.**



a1 – AOS at station 1
a2- AOS at station 2
b1- LOS at station 1
b2- LOS at station 2

Fig 1.2 Station Clash

The other clash situation arises when two or more satellites controlled by the same controller require support at the same time. The controller can't give commands to two satellites at the same time and hence the schedule must decide which satellite is to be supported. The situation is called a **controller clash.**

1.4 Constraints that affect task scheduling.

The problem we wish to tackle in this investigation is the optimum allotment of tasks to open 'windows'. Here an open window develops due to the interaction of a satellite and a ground station as the satellite 'passes over' that station. This is called 'visibility window' as the satellite is visible to the ground station during this time interval. The scheduling of operations in an open window is a difficult task. The complexity arises due to the nature and types of *constraints* and *interrelationships* among different operations.

It is difficult to find a direct match of this problem to conventional machine scheduling problems described in the literature as flow, job or open shop. Various types of special constraints exist in satellite support scheduling.

1.4.1 Constraints induced due to the nature of satellite operations.

- Some operations should be scheduled in requested visibility. These include the requested payload operations.
- Some operations require other operations as prerequisites and/or post-requisites.
- A special category of operations is 'routine' operations. These operations are performed on routine basis. However, these operations have preferred and feasible passes defined for them.
- The next category is the 'special request' operations. These operations come very rarely but when present, these must be done in the requested pass. This may force some other operations to be postponed or preponed.

1.4.2 Constraints induced due to ground stations:

- An operation can only be performed over a ground station if the station has the required capability to perform that specific operation.
- A station can support a satellite only if it has capability to support that specific satellite.
- Each station has its own *reconfiguration time* that is a big constraint.

1.4.3 Satellite induced constraints:

- Satellites have their own duties and priorities defined. For example, IRS-1C is a payload satellite while SROSS is meant for scientific operations.

1.4.4 Controller caused constraints:

- In performing a control operation, all control signals are given by a human controller. A single controller may control more than one satellite. However, the controller can't give commands simultaneously to two different satellites, and hence the previously described controller clash situation arises.

1.4.5 Constraints induced due to visibility scenarios:

- A scenario of visibilities can be a complex set with many different satellites visible at the same time at a given ground station. Other possibility is that same satellite is simultaneously visible from two different ground stations. These are complex clash situations that have to be resolved.

1.5 Thesis Objective:

1. To develop an objective function for scheduling support of TTC type of operations that will tackle the issue of dynamic priority by taking into consideration various factors discussed with ISRO.
2. To check the performance of different metaheuristic algorithms such as Tabu Search, Simulated Annealing and Greedy Search on the problem of clash resolution of satellite visibilities over a station in the satellite scheduling system.
3. To implement the algorithms on sample problem and compare the results on the basis of following parameters.
    - Final solution obtained.
    - Time taken
    - Consistency of algorithm.
4. Compare these results obtained with the results already obtained by applying GA to the same situation.

5. To produce a weekly schedule of clash resolved support using the best optimisation method.

1.2 Thesis Organization:

Chapter 1 provides an introduction to the concept of satellites and its motion. This is followed by the problem overview and the literature review in Chapter 2. Chapter 3 deals with the development of the objective function to help resolve visibility clashes for TTC type of operations and the mathematical formulation of the problem given by Kumar and Bagchi [3]. Chapter 4 gives an alternate approach proposed in this study to solve the problem using the Greedy search heuristic, its implementation, and the results. Chapter 5 discusses a method to solve the problem using Tabu Search, its implementation, and the results. In Chapter 6 we have presented the Simulated Annealing (SA) approach for the same problem. Also, we have optimised the critical parameters in SA. In the same chapter, we discuss the implementation and the results obtained by using SA. In Chapter 7 we discuss the approach to solve the same problem using GA as given by Kumar and Bagchi [3]. In Chapter 8 we compare the first three methods. Then we compare the results and computation time of these three algorithms with GA. The JAVA programs for these algorithms are given in Appendices.

Chapter 2

Problem Overview and Literature Review

2.1 Problem overview:

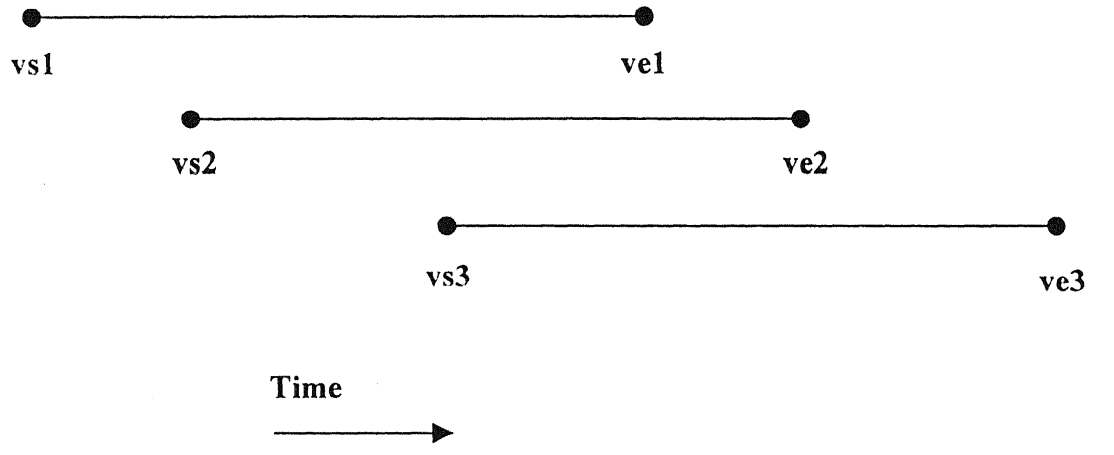
A satellite-scheduling problem is the problem of assigning start times, end times and resources to a set of constrained tasks, such that multiple mission objectives are satisfied. The mission tasks and resources are subject to numerous physical, geometric and operational constraints. Satellite mission planning problem differs from traditional scheduling problem in several ways [6]. Perhaps the most obvious difference is the fact that some of the resources (i.e. satellites) are orbiting the earth. This places an additional set of constraints on when a task can be executed. Also, satellite mission planning problems may involve task start time preferences, consumable resources, periodic tasks, variable duration tasks, and tasks that can be pre-empted. In addition these problems are usually over constrained.

Kumar and Bagchi [3] have described the problem addressed here in terms of scheduling terminology as,

- The opportunity window is open for a specified period within which the resources must be allocated so as to maximise the total profit contributed.
- Products are perishable, have no stock/salvage value, and consumed as produced.
- Each product has its own marketing opportunity.
- Market windows of products may clash, forcing production of only one at one time.
- If a product is produced, a specified minimum quantity of it must be produced.
- Once started, production can't be interrupted.
- For each product there is a maximum quantity that must be produced.
- A finite reconfiguration time is required to switch the machine from one task to another.
- The machine itself is available for a specified period.
- An arbitrary profit function determines how much profit each product contributes.

A simple version of this problem resembles parallel machine scheduling. The goal is to find an optimal set of start times, such that the capacities, availability and time limit constraints are met and as objective function is optimised.

A visibility clash occurs when one or more satellites visible at a ground station have either overlapping visibilities or their respective AOS (Acquisition of signal) and LOS (Loss of signal) difference is less than the station reconfiguration time. The problem is to find the optimum support schedule that maximises the returns (or the total value generated) from supports. A clashing situation is difficult as the patterns in which satellites moves are enormous. The solution search space is large and discontinuous. The problem is NP-complete. [3]



vsi =visibility start time of ith satellite.

vei = visibility end time of ith satellite

Fig 2.1 Visibility Clash

2.2 Literature Review

In India, there is not much work going on over the satellite-scheduling problem. Some researchers such as Kumar and Bagchi [3] have worked on this problem. In ISRO, some work has been done for satellite scheduling and clash resolution. They have developed software called MSS (multisatellite scheduling software) which generates the operations schedule and allocates TTC network resources for different satellites supported at Spacecraft Control Centre (SCC). Satellite specific constraints, ground station configurations, satellite priorities, mission requirements and priorities of certain payload and special operations, as well as visibility conflicts are taken into consideration while generating operations schedule. [17]

The clash resolution logic given by ISTRAC is as follows [17]

The CLASH module considers the sorted visibility records of all the spacecrafts for one day and these records are divided orbit-wise into different sets. The records in a set are also arranged station-wise and for each station the records that do not have any visibility clash are determined. Such records are called confirmed records i.e., the support for the spacecraft is confirmed. The weights of all the confirmed records are computed. These weights have a significant role to play at the time of clash resolution. The Confirmed Record Weightage (CREW) is a function of maximum elevation, no. of operations, station priority and operation priority.

CREW = f (Maximum elevation , no. of operations, station priority, operation priority)

For each ground station among the available clash records, clash weights are derived based on the spacecraft priority. The clash weightage (CW) is a function of maximum elevation, spacecraft priority, no. of operations, operations importance, exclusive pass, minimum operations met, next pass visibility, previous pass visibility, last pass of the day, CREW .

CW = g (Maximum elevation, Spacecraft priority, No. of operations, operations importance, exclusive pass, minimum operations met, next pass visibility, previous pass visibility, last pass of the day, CREW).

The formulation for deriving CREW and CW has been mathematically modeled by taking in to account the function variables and spacecraft control parameters. Having derived clash weightages for all the clash records, the combination is selected for which the clash weightage is maximum. i.e. maximize the weightage function (WF) defined below:

$$WF = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{CWij}{2 * j}$$

m = no. of spacecrafts.
n= no. of stations for spacecraft "i"

Pickup the i and j for which the above weightage function is maximum
This procedure identifies one optimum combination of records for which the visibility clash is resolved, the remaining records in the set are tagged as ' NO SUPPORT'. The same process is carried out for the entire data considered for scheduling.

Kumar and Bagchi [3] have worked on this problem and have evolved with a GA (Genetic Algorithm) based solution to the visibility clash resolution problem. In the paper presented (*OR-2000 XXXIII Annual Convention of the OR Society of India, Ahmedabad, Operations* ***management*** *conference 4, Madras*), they have described a new method for optimising ground station support to LEO (Low Earth Orbiting) satellites. The methodology invokes Darwin's principles of natural evolution to handle objective functions that are arbitrarily shaped and constraints that do not possess tractable structure. They have indicated up to 60% improvement over manually developed support plans. Their method can also help find optimal location for ground stations and support capability deployment in widely diverse scenarios.

Wolfe and Sorensen [19] have described three different approaches to assigning tasks to earth observing satellites. A fast and simple priority dispatch method is described and shown to produce acceptable schedules most of the times. A look ahead algorithm is then introduced that outperforms the dispatcher by about 12% with only a small increase in run time. The third method they used is GA. The genetic approach uses two additional binary variables, one to allow the dispatcher to occasionally skip the job in the queue and another to allow the dispatcher to occasionally allocate the worst

position to the job. These variables are included in the recombination step in natural way. They implemented the algorithms on "window constrained packing problem" which is similar to the station clash resolution problem.

The Generic Resource, Event and Activity Scheduler (GREAS) Application Framework (AF) is a robust collection of software components for building satellite mission planning applications. [9] The GREAS Scheduling Framework utilizes a constraint programming approach for solving these satellite mission-planning problems. Constraint programming uses information contained in the problem to "prune" the search space, rapidly identifying feasible solutions. It is ideally suited for operational problems, which require fast, feasible answers. Constraint programming methodology allows the natural expression of very complex relationships, including logical expressions. The Scheduling Framework builds upon the constraint programming capabilities of commercial software ILOG Solver and ILOG Scheduler to meet the challenges of solving satellite mission planning problems.

The Space Telescope Science Institute [10] has developed a general constraint based scheduling framework for scheduling operations of NASA's Hubble Space Telescope. The approach is to treat the problem as a constrained optimisation problem. It uses a heuristic repair-based scheduling technique called multistart stochastic repair. This technique first makes a trial assignment (or initial guess) of activities to times, based on heuristics. Such a schedule will generally have temporal or other constraint violations, as well as resource capacity overloads. Then they apply heuristic repair techniques to try to eliminate constraint violations, until either a pre-established level of effort has been expended or there are no conflicts left. The third step is to eliminate conflicts by removing any activities with constraint violations, or by relaxing constraints, until a feasible schedule remains. The heuristics employed in are stochastic.

Agnese and Brousse [1] apply depth first, branch and bound methods and greedy search to set task allocation to satellite visibility windows. The exact methods such as depth first, branch and bound etc. have capability of providing optimal solutions at the cost of computational time.

Bottcher, Jahn, Lutz and Werner [4] have worked on basic design problems of LEO/ICO-based networks. In first part of the paper titled "Analysis of basic system parameters of communication networks based on low earth orbit satellites", the estimates for the necessary number of satellites, orbits and number of communication channels per satellite are derived. Since the latter is a crucial quantity, the number of communication links and channels per link are derived with a more elaborate model in the second part of the paper. This includes the radio links from the satellites to mobile users and to gateways, as well as intersatellite links and terrestrial lines. They also introduce a formal model for LEO/ICO- based networks and propose a method for the evaluation of link capacities, given the network topology and the traffic requirements. As an example, two constellations are investigated in detail. One of these constellations is the IRIDIUM system proposed by Motorola.

Kennedy, Gregory and William [11] have developed a prototype expert system for scheduling satellite supports for satellites in Global Positioning System. The prototype used Texas Instruments' Personal Consultant Plus software and incorporated a number of unique search routines and graphics. This is described in detail below.

Scheduling communications contact between satellites and tracking stations (satellite support) is a complex manual process that requires specialized knowledge and expertise. A satellite support occurs when a line-of-sight communication link is established between the satellite and a ground remote tracking station (RTS) to transfer electronics data down. Each RTS can communicate with one satellite at a time. The length of time available for a satellite support (support window) varies from a few minutes for a low-earth orbit satellite (200 KM) to continuously for a geosynchronous satellite (36,000 KM) visible at an RTS. In geosynchronous orbit, the satellite's location remains fixed with respect to the surface of the earth.

The planner analyst (PAs) determines the number of support required for each satellite and the preferred schedule. The range schedulers must schedule the RTSs to support 50 to 100 satellites. The scheduling objectives are to minimise the total number of supports required to accomplish each satellites mission and to provide RTS schedulers as much as possible.

The Air Force Satellite Control Facility at Onizuka Air Force Station, California, requested to the authors to examine the feasibility of using expert systems (ESs) to assist the planner analysts. They used ES technology to develop a prototype advisor system to help planner analysts schedule satellite supports. They selected the Global Positioning System (GPS) satellite program for prototype development for two major reasons: first, each of the 20 satellites in the GPS system requires separate satellite support scheduling, and second, the GPS program if of medium scheduling difficulty because the periods GPS satellites are "visible" to an RTS are shorter than for geosynchronous satellites and longer than for low-earth-orbit satellites.

Their objective was to demonstrate that, using ES technology for scheduling satellite supports was feasible and cost effective. Based on the success of this prototype, several ES projects have been initiated by the Air Force and supporting contractors.

The satellite support scheduling Process

Scheduling satellite supports is a three way process: First, satellite users send requests to the Mission Control Complexes (MCC) for a satellite to perform particular task at a certain time. There are several MCCs, each responsible for the satellites performing a particular mission. Each satellite is assigned a unique operations number, for example 5111.

Second, PAs prepare seven-day schedule of the supports required for each satellite (referred to as Program Action Plans or PAPs). For each satellite, PAs prepare cover sheets that outline the support requirements for the seven-day period. Next, PAs prepare satellite visibility charts (fig), which indicate all times each satellite is visible at each RTS for the first 24- hour period. Finally, they generate PAPs which include the following information: satellite vehicle operations number, planning period, date of each support, start/stop times of the support window, contact duration time, visible RTSs throughout the support window, type of support, and special support requirements (for example, unique equipment).

Third, each satellite's PAP is transmitted to the Range Control Centre, an office where schedulers build seven-day and 14 day schedules assigning each satellite support to a visible RTS.

VISIBILITIES FOR MONDAY -- 16 JUN 86

SATELLITE VEHICLE: 5111

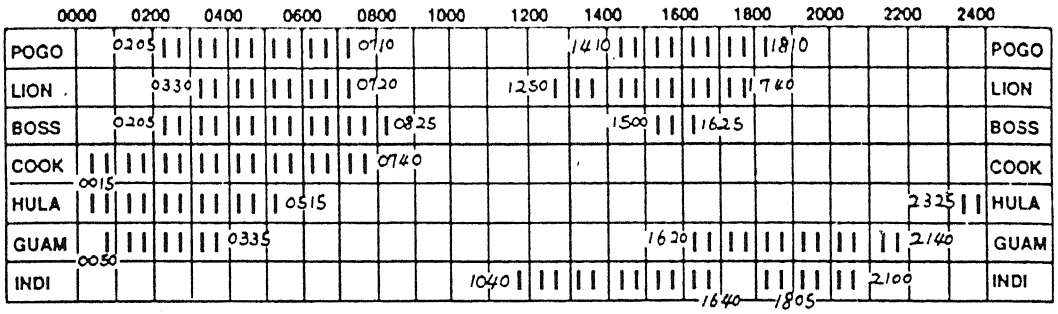


Figure 1: A sample planner analyst's visibility chart which indicates the times that satellite #5111 is visible at each of the seven RTSs for a 24-hour PAP period. Each satellite is assigned a unique tape color pattern and rolls of tape with that color pattern are used by the PA throughout the support scheduling process to show the duration of the support windows. The exact start and end times of visibility are handwritten at each end of the tape strips.

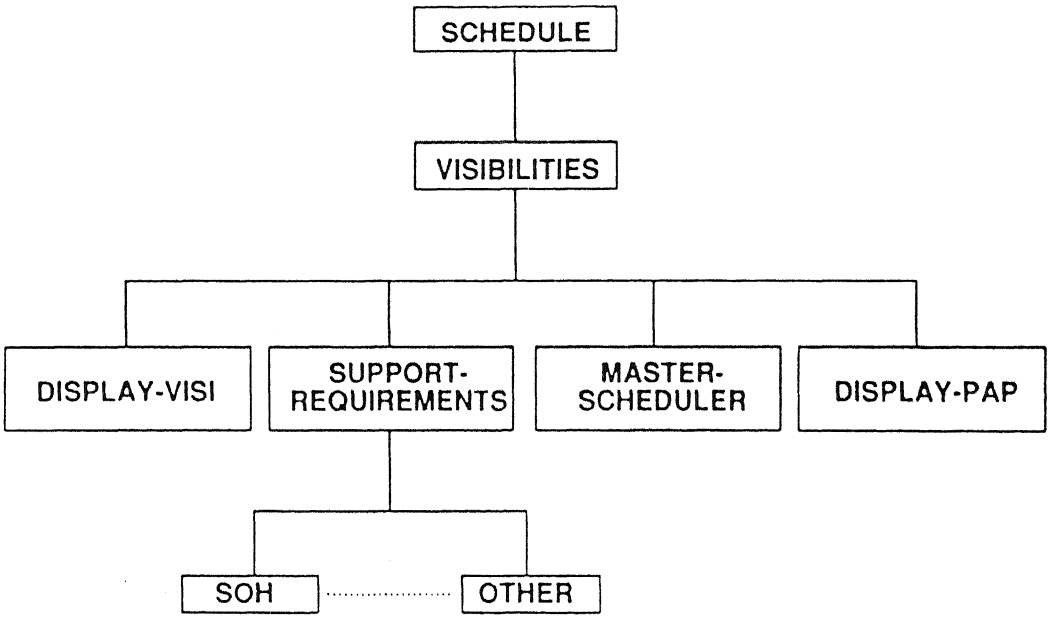
Why Use Expert System Technology?

Expert systems are appropriate for three reasons: (1) expert knowledge is important in developing PAPs, (2) the scheduling process is very rule oriented, and (3) automating the PAPing process should save money.

Only certified PAs prepare the PAPs. Each PA is trained on the job for three to six months and must pass qualifying examinations to become certified. Because of the complexity of the process and the importance of the satellite supports, each PAP is checked by a second certified PA for accuracy and completeness.

In preparing PAPs, PAs use several documents and also rely on their extensive experience and expertise. They refer to computer printouts of ephemeris data, which provide RTS visibilities and orbit geometries for each satellite vehicle. Various documents (test operations instructions and memograms) provide the rules for each type of support. These heuristic methods and the scheduling rules can be modelled using ES technology.

Finally, managers at the Air Force Satellite Control Facility believe that there is a clear economic incentive to automate the process. Currently the process takes 20 hours per week per satellite, and 20 satellites are planned for the GPS system. An ES could reduce the 400 hours per week by an order of magnitude.



re 2: The prototype system design uses a hierarchical frame knowledge representation s .cture. Each frame represents a subtask to be performed and contains the procedural and heuristic rules to perform each task. When a frame is evaluated it also has access to knowledge contained in parent frames. The knowledge representation graph is searched in a depth first manner [Nilsson 1980].

Knowledge Acquisition

Acquiring knowledge about the existing planning process was critical to their work developing a prototype system. They asked the Air Force Satellite Control Facility to assign to the project a PA expert who was very interested in it. After extensive planning, the knowledge engineers made two one-week site visits. Afterwards, the expert reviewed transcripts of the knowledge they had acquired during our visits. They continued to consult with the expert by phone each week throughout the research.

Prototype System Design

Based on an extensive trade-off analysis of ES shells, they selected LISP-based Personal Consultants Plus (PC Plus). PC Plus [Texas Instruments 1986] provides for forward and backward chaining, frames and rules, explanations, user-defined functions, and the graphics interfaces necessary to develop the PAP forms.

(Figure 2) shows the knowledge representation scheme. For each of the boxes shown in figure, there is frame that has unique rules, parent frames, goals, user prompts, and parameters. The ES begins the consultations in the schedule frame and requests the user to identify the satellite vehicle and the PAP period date. The ES then computes the remaining dates.

The visibilities frame is evaluated next. The ES transfers the appropriate visibility data from an external data disk into the knowledge base. Next the display-vis frame is evaluated. The ES asks the user whether he or she wants to display the satellite visibility charts for any day during the PAP week.

The support-requirements frame is then evaluated to determine the required support (for example, satellite state of health (SOH)). Next, each applicable support frame is evaluated. Each of the 18 support frames contains the unique parameters and rules necessary to determine such PAP information as support start time and duration. The prototype implements SOH supports only but is designed to accommodate all 18 support types. The other 17 support type obtains data on specific satellite subsystems, for example, the battery, the thermal control system, and the rate gyro system.

The ES then evaluates the master-schedules frame, which contains the scheduling parameters and rules. Visibility search routines identify the RTSs that are visible at a particular time. If the satellite is not visible from any RTS at the stated time, these routines search for the next best time that the satellite is visible from one or more RTSs. Each frame evaluation produces the PAP schedule for one support.

When all supporters are scheduled,, the display-PAP frame displays the completed PAP form. At the expert's request, all graphic displays were designed to mirror the current manually generated PAP documents.

User Interface

The user interface is menu-driven. To generate a PAP, the user must answer 10 to 15 questions by the expert system. A sample question is

GPS - PROGRAM ACTION PLAN EXPERT SYSTEM ADVISOR

VISIBILITIES FOR MONDAY --- 16 JUN 86

SATELLITE VEHICLE: 5111

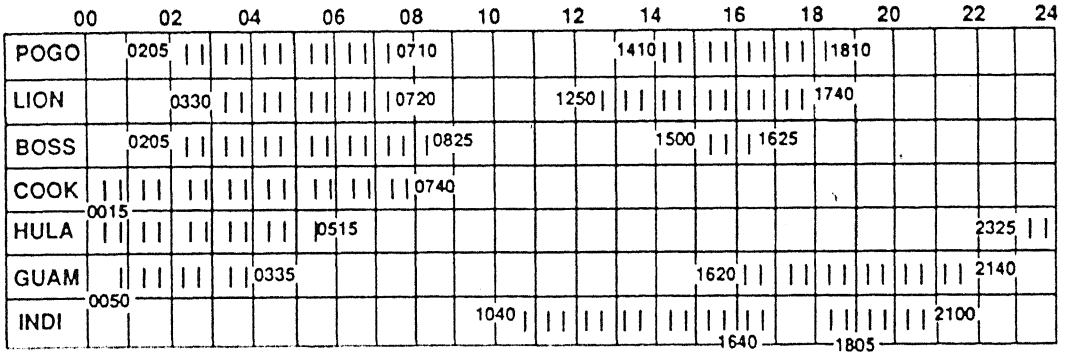


Figure 3: A sample visibility chart developed by the ES shows the same data that was manually generated in Figure 1. The chart shows the visibility times on June 16, 1986 for #5111 at each of the seven RTSs.

GPS - PROGRAM ACTION PLAN EXPERT SYSTEM ADVISOR

PROGRAM ACTION PLAN - GPS											PAGE 1 OF 3 PAGES	
OPERATIONS NUMBER, 5111			EFFECTIVE DATE 16 JUN 86 0000Z THRU 22 JUN 86 2400Z									
DATE	TIME		DUR	STATION DESIRED							PURPOSE OF PASS	SUPPORT REQ'MNTS
	START	STOP	MIN	POGO	LION	BOSS	COOK	HULA	GUAM	INDI		
16 JUN	0320	0400	10	X		X	X	X			SOH	NONE
16 JUN	0745	0825	10			X					SOH	NONE

Figure 4: A sample Program Action Plan form developed by the expert system. The expert system scheduled the same remote tracking stations as the planner analyst.

Enter the operations number of the vehicle to be PAPed:

5111

5112

..9763

Results

Figure 3 is a sample ES visibility chart for satellite #5111 on June 16, 1986. This chart shows the same visibility data as the manually developed chart in figure 1. The ES saves the PA 30 minutes per satellite per week.

A sample of a completed PAP is shown in figure 4. Given the last scheduled support time from the user (June 15 at 2200 hours), the ES determined that a support was required once every six hours and that the best time window for the next support occurs at 0320-0400 (Figure 4). Then the ES invoked the visibility search routines to determine if vehicle # 5111 was visible from any RTS during this time window (see the visibility chart in figure 3). It was visible from Pogo, Boss, Cook and Hula(figure 4). Allowing the maximum time permitted between supports (six hours) , the next support should occure at 0920-1000 hours on june 16, 1986. This choice minimizes the total number of supports required. However, the visibility search routines determined # 5111 was not visible at any RTS at that time. As a result, the search routines found the support could be scheduled and still meet the six-hour constraint (Figure 4)

Conclusion:

Because of this research and prototype effort, the Air Force and its supporting aerospace contractors have increased their efforts to apply expert systems to scheduling satellite supports. Using a similar design, IBM Federal Systems Division is developing a full scale prototype ES called REGE (for Request Generator) for the GPS PAPing process. REGE currently schedules five support types. By late 1988, IBM plans to have an operational version of REGE that will schedule 12 to 13 support types. When fully implemented for all 18 support types, REGE will reduce the PAP generation time by an order of magnitude.

The success of this prototype demonstrates that ES technology can successfully automate the PAPing process. ESs are ideal for problems for which the solutions are

obtained using experts' heuristics and rules. However, for problems as complex as GPS PAPing process, extensive prototype programs are required to develop operational expert systems. Knowledge engineering offers an appropriate methodology and the required software development tools.

After going through the literature available in satellite operations scheduling area, in the next chapter we present the development of an evaluation function for TTS operations scheduling.

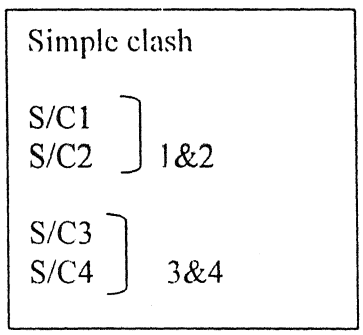
Chapter 3
Derivation of the Objective Function to Help Resolve Visibility Clashes.

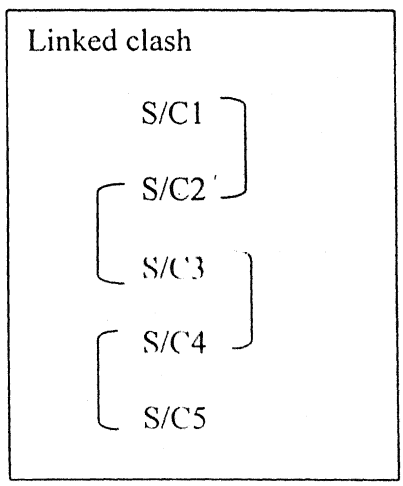
A "visibility clash" occurs when one or more LEO satellites visible at a ground station have either overlapping visibilities or their respective LOS (Loss of Signal) and AOS (Acquisition of Signal) difference is less than the station reconfiguration time. The problem is to find the optimum support schedule, that maximizes the returns (or total value generated) from supports provided. This chapter proposes two separate objective functions to help resolve visibility clashes optimally.

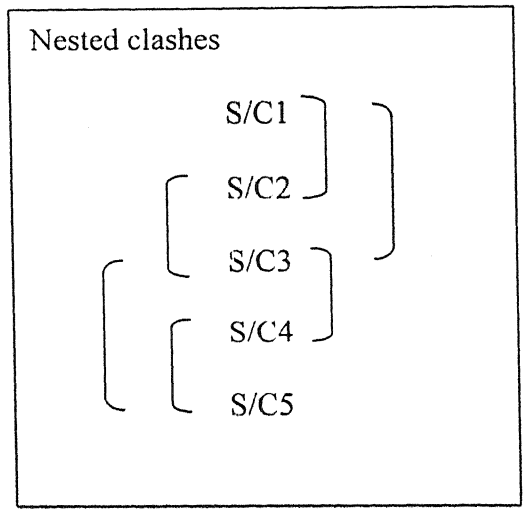
A clashing situation is difficult to precisely specify, as the patterns in which satellites moves are enormous. A clashing situation as envisaged by ISTRAC (ISRO Telemetry, Tracking and Command Network) can have any of possible combinations of overlaps as shown in the following figures.

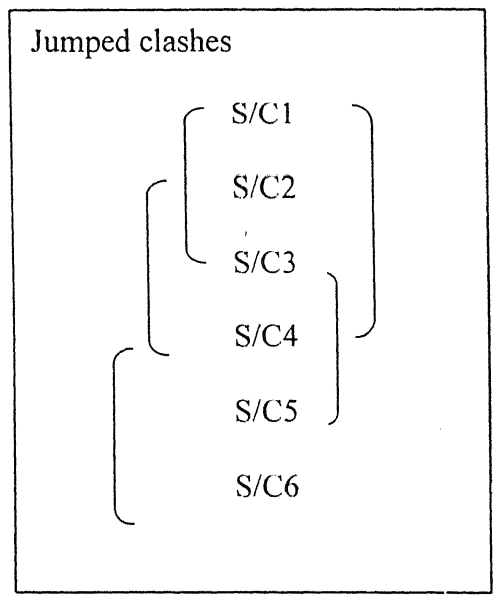
3.1 Possible Visibility Clash Topologies:

Fig 3.1






Nested clashes
S/C1
S/C2
S/C3
S/C4
S/C5


Jumped clashes
S/C1
S/C2
S/C3
S/C4
S/C5
S/C6

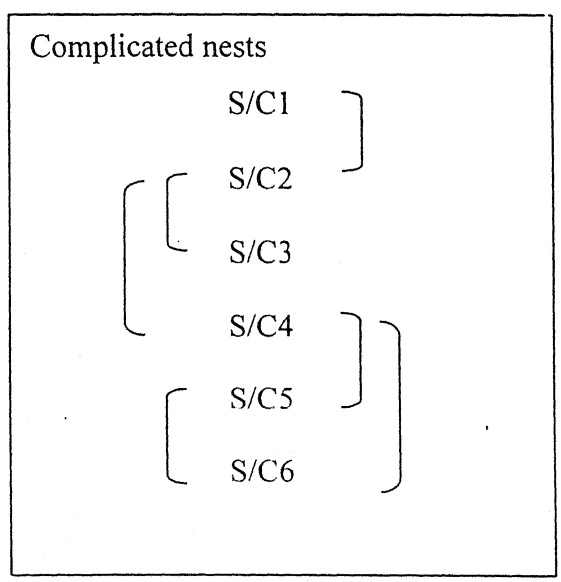
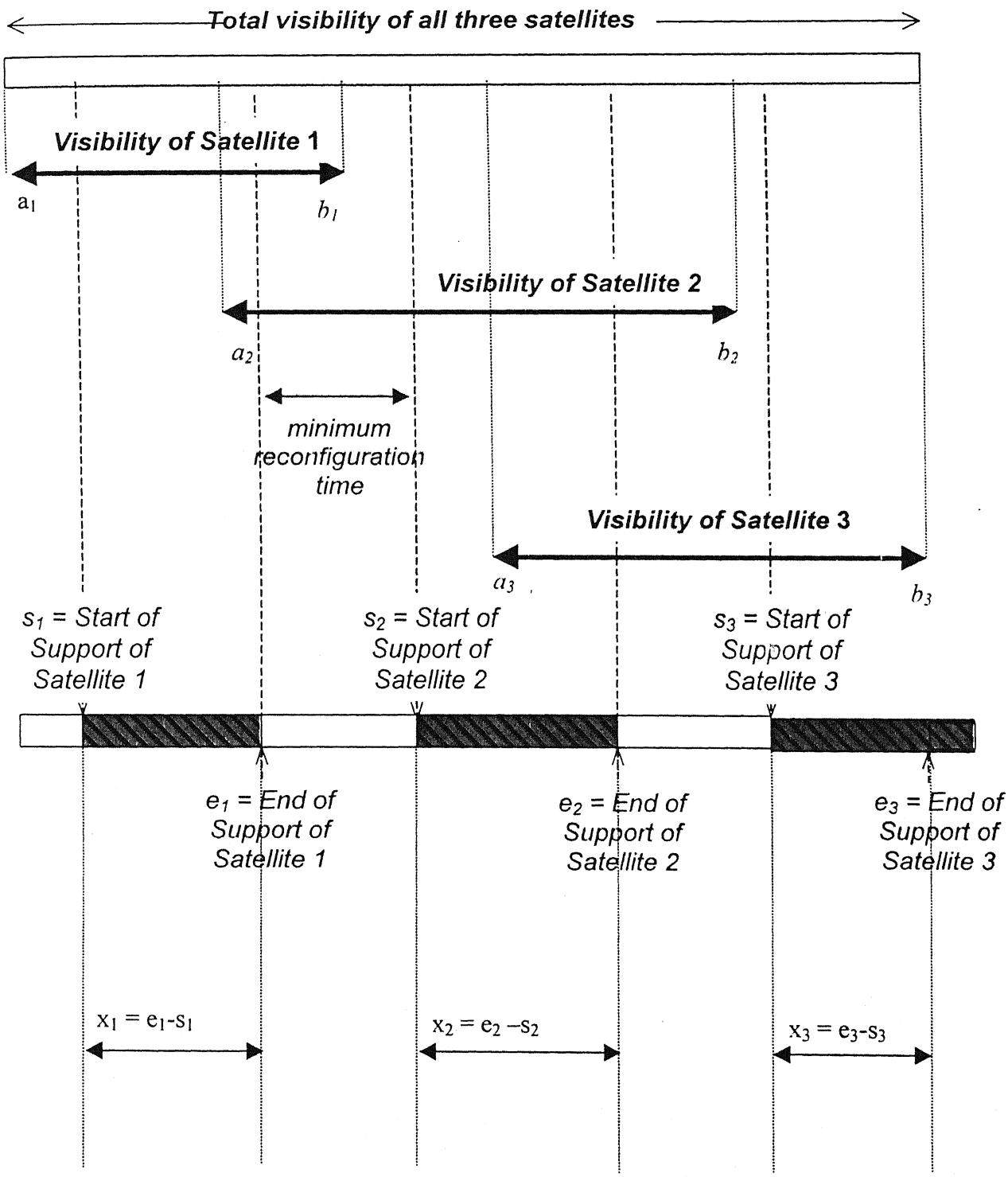

Complicated nests
S/C1
S/C2
S/C3
S/C4
S/C5
S/C6

Fig 3.2 NOTATION REPRESENTATION OF A CLASH SITUATION



3.2 The Notations used (Fig 3.2) in the Mathematical Model of a typical clash resolution problem (Kumar, Bagchi 2000)

V= Difference of first clashing AOS and last LOS.

I = Total number of visibility clashes

$i = 0.....I.$

P= Value function, to be maximized (optimised)

a_i= Start of visibility (AOS) of satellite i.

b_i= End of visibility (LOS) of satellite i.

s_i= Start of support of satellite i.

e_i= End of support of satellite i.

min = Minimum time necessary for a support.

max = Maximum time allowable for support.

r= Reconfiguration time.

v_i= Value contributed to P per unit time by supporting satellite i.

x_i= Support duration for visibility of satellite i.

t_i = Reconfiguration time is added to end of support of previous supported satellite. t_i is a binary variable which reflects whether satellite i is supported or not. If t_i =1 it implies that satellite i was supported while t_i =0 implies that the satellite was not supported. Thus,

t_i is defines as binary variable such that:

if $x_i = 0$; $t_{i+1}=0$

if $x_I>0$; $t_{i+1}=1$

3.3 An Evaluation Function for TTC Support Value Generation

It is known that for TTC operations the return per unit support time has direct correlation to the *duration of support*. From discussions held with ISRO experts we derived following points.

- TTC operations require a minimum support of 8 minutes (480 seconds),whenever support is provided.
- Almost all operations can be done in this 8-minute span. Thus the returns obtained in supporting a satellite for minimum support time are high.
- A support of more then 8 minutes is not necessary, but if some additional visibility time is available, support may be extended beyond 8 minutes. Thus, more then 8 minutes support is desired but not required.

Thus we conclude that return function should be designed in such a way there is no return if a satellite is supported for a duration of less then 8 minutes but the return per unit time is high if support is exactly 8 minutes. However, the return per unit time diminishes after 8 minutes. We may explain it as follows. If a satellite is supported for 10 minutes, the returns obtained are more for the "8th minute" as compared to the "9th minute", and returns in the "9th minute" are more then returns in the "10th minute".

Considering these factors, an exponentially diminishing return function was designed. This function meets all requirements listed above while incorporating a diminishing rate of return. A simple exponential function can be represented as

$$f=\lambda e^{-\lambda t} \qquad \text{..... (3.3.1)}$$

where t is time measured from origin. This exponentially diminishing function is shown in Figure 3.3.

At t=0, $f=\lambda$, thus λ is the ordinate of the exponential curve at t=0.

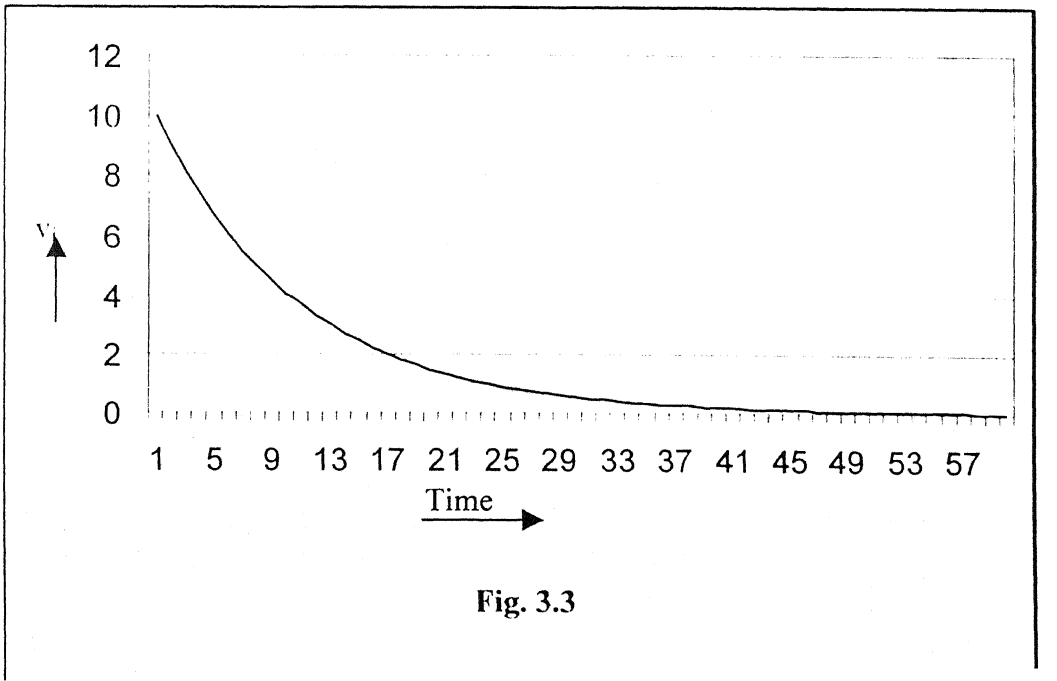


Fig. 3.3

To mould this diminishing function to our requirement, we introduce one more term, B.

$$f=\lambda e^{-\lambda Bt} \qquad \text{.....(3.3.2)}$$

where B is a factor which will determine the actual decay rate of this exponential curve as time t increases.

3.3.1 Determining value of B.

In this problem, as we already indicated, the minimum support needed is 8 minutes. Similarly there is an upper limit also (16 minutes), after which the returns per unit time (or the value generated from support) becomes negligible. Based on the discussions with ISRO, we assumed that the function value might be assumed to reduce to 1% of its initial value (λ) when support time becomes 16 minutes. That is if

t 480 seconds
f=1% of λ

Therefore the value of B may be found as follows.

At t=0, $f=\lambda$
At t=480, f=1% of λ

Hence $\lambda e^{-\lambda*480*B}=\lambda/100$

which gives

$B=0.00958/\lambda$ .....(3.3.3)

At an instant t, the total value of support provided up to t is the area covered between the exponential rate curve and the abscissa. We may determine this total value by integrating the exponential function with the integration going from 0 to t.

Total Value= $\int_0^t \lambda e^{-\lambda tB} dt$,

which gives

Total Value of Support = $(1- \bar{e}^{\lambda tB})/B$ ..... (3.3.4)

But in real life situation, the return function is not exactly exponential rather it has a discontinuous nature. This is shown in Figure 3.4

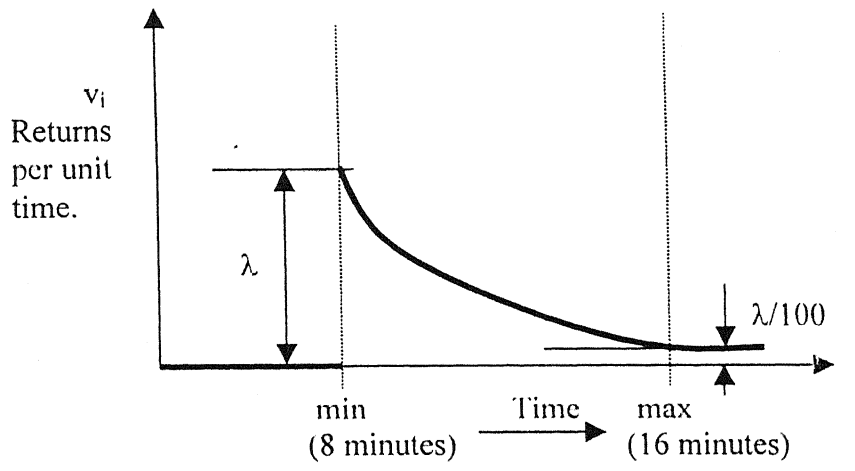


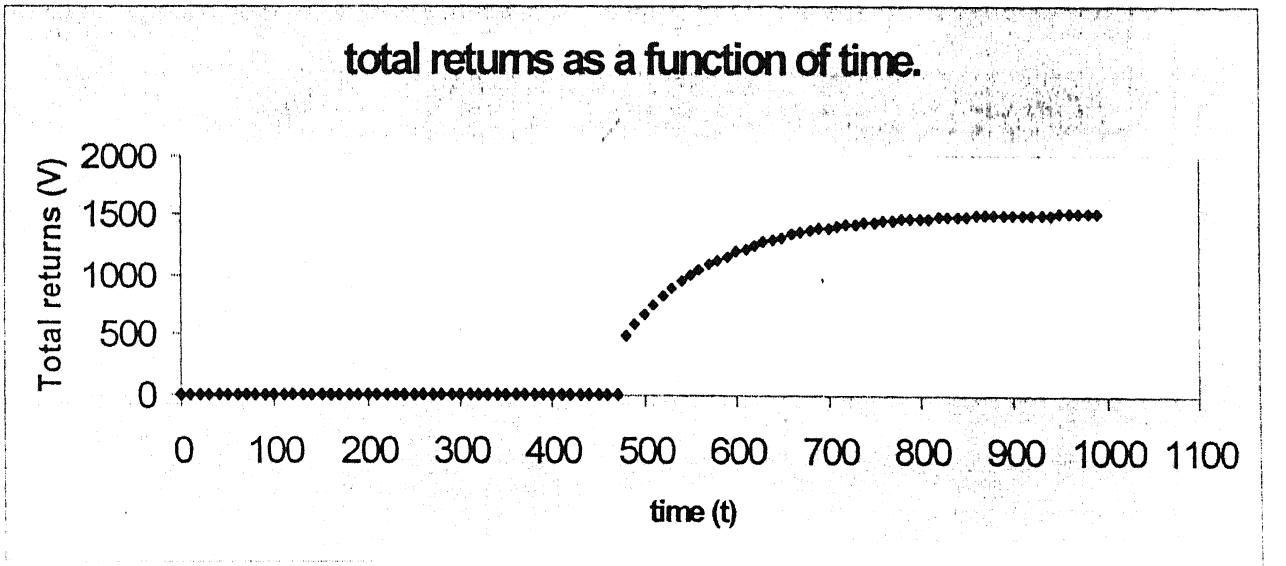
Fig 3.4

Total Return ={A+Exp} **for t≥480;** .....(3.3.5)
Total Return =0 **for t<480;**

Where
A=480*λ
Exp=(1- $\bar{e}^{\lambda t B}$)/B
B=0.00958/λ
t=(Support Duration-480)

The term *A* ensures high returns of the total support provided when a satellite gets a support of 480 seconds.

Total Return as a function of *t* is shown in Figure 3.5



3.3.2 The Derivation of Lambda:

The next challenge in the model is to find the value of λ which in general would be different for every visibility. λ is the rate of return when a satellite is supported for 8 minutes (minimum support for TTC operations). In other words, λ is the height of the exponential rate curve at zero time for the pure exponential function.

The utility of exponential decay logic depends on the derivation of λ. As a result of discussions with ISRO experts, we concluded that λ for a support (a combination of satellite, station and operation) depends on several factors as listed in Table1. These factors each have their own impact on λ. The relative impact of these factors is expressible as shown in terms of objectively stated priority and penalty based on the knowledge of LEO satellite technology.

TABLE 3.1 FACTORS CONTRIBUTING TO SATELLITE SUPPORT

S.No	FACTOR	VALUE	PENALTY
1	Maximum elevation	2	
2	Satellite priority	10	
3	Exclusive pass (orbit over a single station)	8	
4	PL settings (Timer settings)		10
5	Special TTC operations (must be done in that pass)	10	
6	Service (visibility gap)		5
7	Minimum operations in the satellite	6	
8	Exactly 'n' ascending operations per satellite		2
9	Exactly 'm' descending operations per satellite		2
10	Controller constraint		2
11	Ground station constraint	2	

Combining these factors in an equation form gives λ. The potential method of combining these factors reflects their nature and their effect on satellite support. Factors not considered in λ may be used to guide the actual heuristic search for task allocation.

λ=Max (FactorExclusive, FactorCyclic, FactorTimer, FactorSupportGap) +FactorSatellite+FactorElevation. .....(3.3.6)

FactorExclusive, FactorCyclic, FactorTimer and FactorSatellite are set on a scale of 1 to 10. Other factors depend on dynamic parameters. FactorSupportGap depends on the time gap between current visibility and previous supported visibility for the same

satellite. Factor elevation is step function assumed for illustration to be rising from 0 (for elevations up to 5°) to 10 (for elevations >= 60°).

Note that the above expression for λ has two parts. The first part is the maximum of four factors while the other part added to it. It is noticeable that the first part has terms that correspond to the actual satellite operation. FactorSatellite and FactorElevation correspond to satellite concerned and the quality of the visibility window and are independent of the actual operations. Further, the first part has factors that are very important in support decisions; if any one factor among these is high, the visibility should be supported. This requirement is ensured by taking maximum of Factor Exclusive, FactorCyclic, FactorTimer and FactorSupportGap.

3.3.2.1 Effect of Parameters of λ

FactorExclusive: This factor assumes values of either 0 or 10. If a visibility is exclusive in that specific orbit, this factor becomes 10. Otherwise it is 0. Thus if a visibility is exclusive, this factor increases the value of λ so that the likelihood of support for this visibility rises.

FactorCyclic: This factor also takes values of 0 or 10. If a cyclic operation falls in this slot of visibility, this factor becomes 10 otherwise it is 0. Thus if a satellite requires a cyclic operation in this particular visibility, this factor becomes 10 and thus increases the value of λ and hence the likelihood of providing support for this visibility becomes high.

FactorTimer: This factor also takes values of 0 or 10. If a Timer operation (required by a subsequent payload (PL) operation) falls in this slot of visibility, this factor becomes 10, otherwise it is 0. Thus by giving 'Timer Operation' a value of 10 we ensure that the subsequent PL operation is serviced as timer setting is prerequisite for a PL operation on certain satellites.

FactorSupportGap: Generally speaking, a satellite has to be supported throughout the day. An important consideration in satellite support scheduling is to keep the gap between supports small. FactorSupportGap ensures that a satellite will have an increased likelihood of support after a certain time gap. The value of this factor depends on the time gap between current visibility and previous supported visibility for the same satellite. As this gap increases, FactorSupportGap also builds up. After

some gap this will build to such a high value that it will exceed any other factor in value. Thus such a a satellite will receive a support.

FactorSatellite: This factor is a satellite-dependent factor and is independent of operations and visibility. This factor is additive term in λ. This factor indicates that when a conflict develops, more important satellite (as indicated by network management) should have a higher probability of getting a support.

FactorElevation: The FactorElevation has a step function. This factor also is additive in λ. This indicates that a higher elevation visibility is more suitable for doing an operation as compared to low elevation visibility.

3.4 Evaluation function for Payload Operations Support

For Payload operations, the return per unit time is constant; hence the evaluation function becomes linear. It is then simply the product of the support time (x_i) and returns per unit time (v_i), beyond any minimum support required.

$P = \sum_{\text{over all } i}\{v_i * x_i\}$ ..... (3.4.1)

The function is shown in fig. 3.6

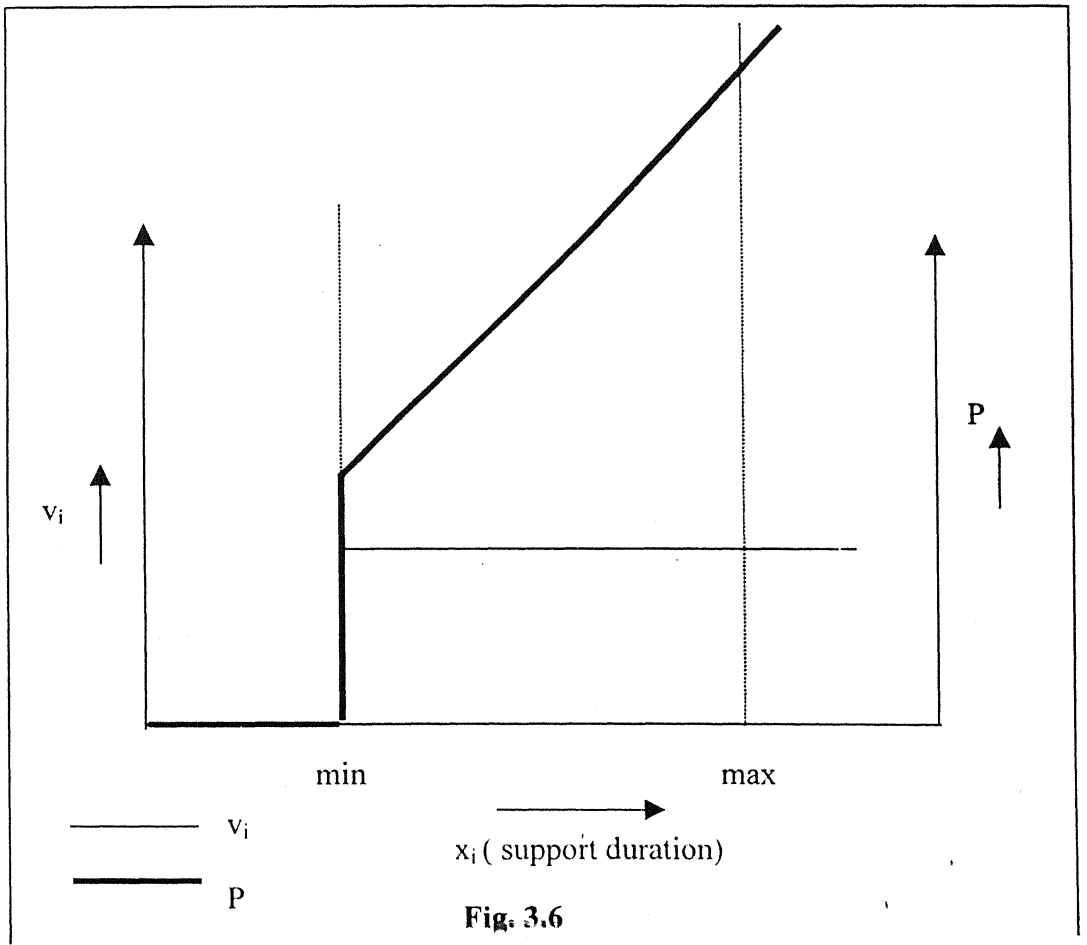


Fig. 3.6

3.5 The Mathematical Model for TTC Support

Maximize,

$$P = \sum_{\text{over all i}} \{Ai + ((1 - \bar{e}^{\lambda itB})/B)\} \quad \ldots\ldots \quad (3.5.1)$$

Subject To:

i) Start of support of satellite '*i*' must be at AOS_i or later and it should be equal to or less then LOS_i.

$$b_i \geq s_i \geq a_i \quad \ldots\ldots\ldots \quad (3.5.2)$$

ii) End of support of satellite '*i*' must be at LOS_i *and it should be equal to or more than* AOS_i.

$$a_i \leq e_i \leq b_i \quad \ldots\ldots\ldots \quad (3.5.3)$$

iii) Reconfiguration allowance.

$$s_i \geq \{ \underset{k=(i-1)\ldots 1}{Maximum} (e_k \times t_k) \} + r \quad \ldots\ldots \quad (3.5.4)$$

iv) Duration of support of satellite *i*.

$$x_i = e_i - s_i \quad \ldots\ldots\ldots \quad (3.5.5)$$

v) Constraint for minimum time of support. This may be either 0 or greater than min

$$x_i = 0 \ \ or \ \ x_i > min \quad \ldots\ldots\ldots \quad (3.5.6)$$

vi) Constraint for maximum time of support. This should be less than *max*.

$$x_i < max \quad \ldots\ldots\ldots \quad (3.5.7)$$

vii) Non negativity constraint

$$s_I,\ e_I > 0; \quad x_i \geq 0\ ;$$

The objective function 3.5.1 and the presence of constraint 3.5.6 makes the problem non-linear.

In the above formulation, for convenience and without loss of generality we made the assumption that reconfiguration timings are constant and have same value 'r' for all set of ground stations and satellites.

The above objective functions, (3.3.5) and (3.4.1) will be used in the subsequent chapters to actually resolve visibility clashes optimally. (3.3.5) models the objective for TTC visibility support while (3.4.1) models the objective for a PL support.

Chapter 4.
Greedy search Approach for Optimal Resolution of Visibility Clash

A Greedy Search or Neighbourhood Search is a local search algorithm, which is based on iterative improvement. The application of a greedy search algorithm presupposes the definition of configurations, a cost function and a generation mechanism, i.e. a simple prescription to generate a *transition* from a configuration to another one by small perturbation. "Configuration" means a set of values of decision variables. The major disadvantage of this algorithm is that it ends up in a local optimum solution. Therefore, in combinatorial optimisation problems, some variation of greedy search is used to skip out from local optima. The major advantages and disadvantages of this algorithm are discussed in more detail in section 4.1.1 of this chapter.
In this chapter we refer closely to the mathematical formulation of a sample global optimisation problem encountered in LEO satellite support scheduling.

4.1 Approach using Greedy search

4.1.1 Greedy Search:

The simplest type of search in optimization is local, greedy search, in which we start with a solution X. At each step of the search, we choose the neighbor of X, $X(n)_l$ such that $X(n)_l$ is lower than the cost of any other neighbor and is also strictly smaller (or larger, if a maximum rather than minimum is desired) than any previously-encountered cost. The algorithm stops when no such neighbor exists. This approach is also known as hill climbing (when searching for a maximum) or descent (when searching for a minimum). In the case of a complicated objective function with multiple modes, this type of search is likely to reach a local optimal point, so one common variation is to repeat the procedure with a number of starting positions and take the best result.
The disadvantages of the Greedy Search method are as follows (ref).

- By definition, Greedy Search algorithm terminates in a local optimum and there is generally no information as to the amount by which this local optimum deviates from a global optimum.
- The obtained local optimum depends on the initial configuration or starting point, for the choice of which generally no guidelines are available.

- In general it is not possible to give an upper bound for the computation time of greedy seaarch.

However, this algorithm has the advantage of being broadly applicable: Configurations, a cost function and a generation mechanism are usually easy to define. Besides, though upper bounds for computation times are missing, a single run of an greedy algorithm (for one initial configuration) can on the average be executed in a small amount of computation time.

To avoid some of the aforementioned disadvantages, one might think of a number of common modifications to the basic greedy search.

1. Execution of the algorithm for a large number of initial configurations, say N (e.g. uniformly distributed over the set of configurations R) at the cost of an increase in computation time; for $N \rightarrow \infty$, such an algorithm finds a global optimum with a probability of 1, if only for the fact that a global minimum is encountered as an initial configuration with probability 1 as $N \rightarrow \infty$;
2. (Refinement of 1.) Use of information gained from previous runs of the algorithm to improve the choice of an initial configuration for the next run (this information relates to the structure of the set of configurations);
3. Introduction of a more complex solution generation mechanism (or, equivalently, enlargement of the neighborhoods), in order to be able to 'jump out' of the local optima corresponding to the simple generation mechanism.
4. Acceptance of the non-improving transitions in the cost function in a limited way.

The second and third approaches are problem-dependent and hence their applicability is limited. The first approach is the traditional way to solve combinatorial optimization problems approximately by greedy search.

4.2 Initial Solution Generation Mechanism:

To tackle the satellite clash resolution problem, we have repeated the greedy search procedure with many different starting solutions. In this problem context, if we look at the search space more closely, it would resemble many hills distributed at various locations in the whole search space. The peculiarity of this search space is that these various hills have no continuity between them, and hence while searching, one has to sometimes jump from one hill to another in order to come out of local optimum.

Furthermore, even if we consider a single hill, it may also have more than one peak and hence search becomes even more complex.

Thus in this problem the challenge would be to select sufficiently many different starting solutions so that no part of the search space would remain unexplored. We took the advantage of the fact that, a satellite, if supported, has to be given at least a minimum time of support. Otherwise the satellite is said to be unsupported and if unsupported, the support time is zero. Therefore we generated 2^n-1 different binary combinations of possible support scenarios where n = number of satellites clashing over a station. The combinations generated are simply in accordance with the binary numbers where a '0' represents that the particular satellite is "not supported" and '1' represents that the satellite is "supported" with minimum support allocation. This is explained below with an example.

Suppose 3 satellites are clashing over a station. Let's call them sat1, sat2, sat3. Now we would generate 2^3-1 =7 different binary combinations as shown below.

001, 010, 011, 100, 101, 110, 111.

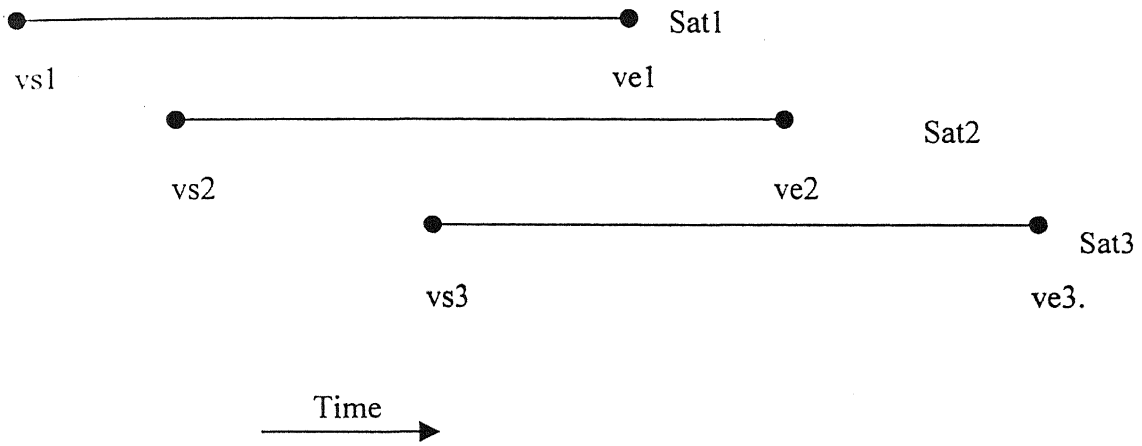
Note that 000 (No support to any satellite) is not an interesting case, hence it is omitted. Now the first binary number, that is 001, indicates that, in the first initial solution generated, sat1, sat2 will not be supported as there is '0' in the first two places of the binary number, and only sat3 will be supported as the binary number has '1' at the third place. Similarly, the second initial solution generated will have sat2 supported and 'no support' for sat1 and sat3. Similarly 7 different initial solutions will be generated with final solution supporting all the three satellites.

One way to further reduce the number of initial solutions is to consider only those initial solutions, in which satellites having maximum weight factors clash. This is because, after experimentally running the algorithm for all the initial solutions we found that, most of the time, the best solution obtained contains the support for the satellites with maximum value of the weigh factors $\{v_i\}$. Hence if we consider only those initial solutions in which the satellites having maximum weight factor clash, we would be able to reduce the computation time without likelihood of loosing the best solution existing.

As already noted, there is one major problem with the greedy algorithm. Even though we would generate one initial solution for one hill in the solution space, some hills have multiple peaks and the greedy algorithm may end up in a local optima. Hence we

may have to consider more initial solutions than 2^n . We will consider this case with an example.

Suppose sat1, sat2 and sat3 are supported in one of the initial solutions(case of '111'). The situation is like this.



Where,

vsi =visibility start time of i^{th} satellite.

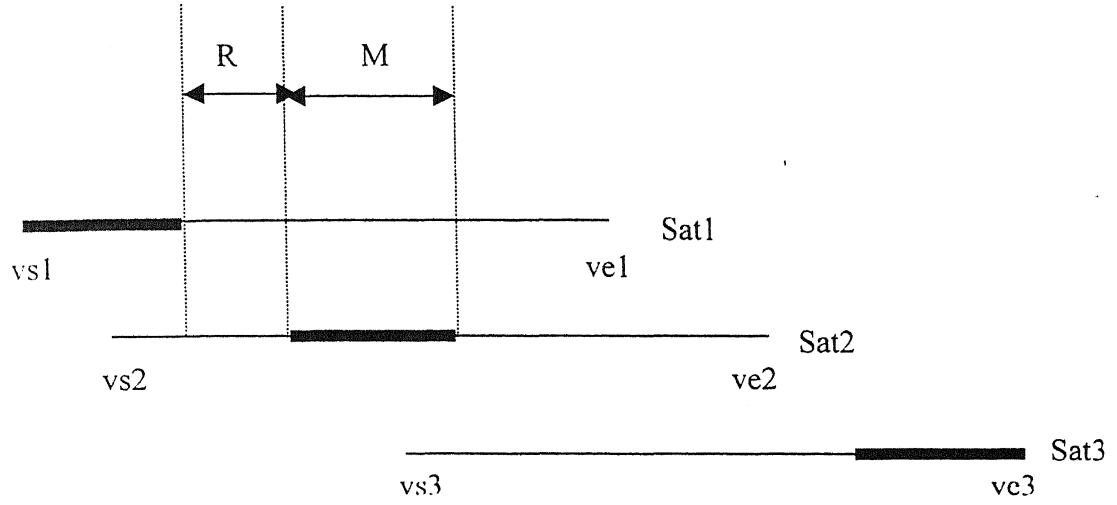
vei = visibility end time of i^{th} satellite

In this situation we have to generate two initial solutions. The first one will have sat2 supported from vs2 side, that is, the minimum support allocation is done from vs2 side and hence,

Allocation start time of sat2 = max((allocation end time of sat1+station reconfiguration time),(visibility start time of sat 2)) ..... (4.2.1)

Allocation end time of sat2 = Allocation start time of sat2+ minimum support time...

.....(4.2.2)



Non-Allocated Window

Allocated Window.

R - Station reconfiguration time.

M – Minimum support time.

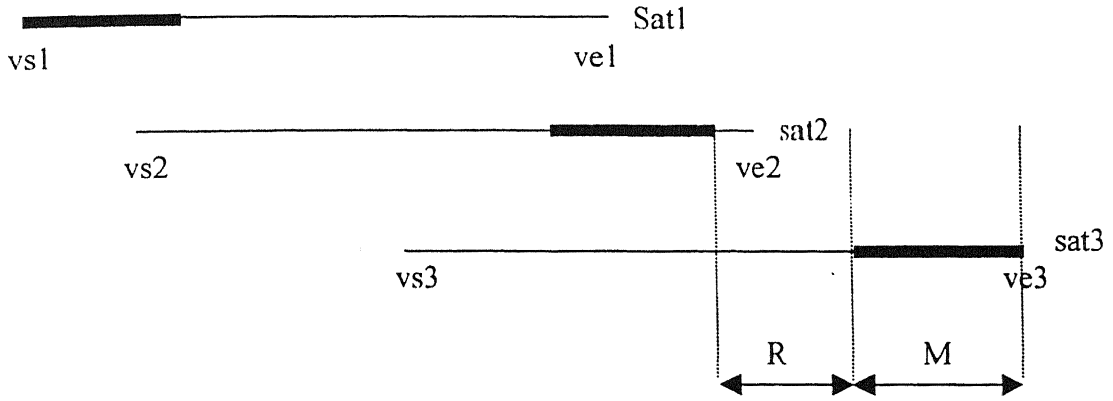
The second situation will be like this- Sat1 and Sat3 will be allocated windows as above and sat2 will be allocated a window from ve2 side. Now the equations become,

Allocation end time of sat2 = min((allocation start time of sat3-station reconfiguration time),(visibility end time of sat 2)) ..... (4.2.3)

Allocation start time of sat2 = Allocation end time of sat 2- minimum support time.

..... (4.2.4)

The figure is shown next.



Non allocated window.

Allocated window.

R – Station reconfiguration time.

M – Minimum support time.

These two initial solutions will both correspond for the '111' situation.

Hence the modified formula for number of initial solutions is,

Number of initial solutions = $2^n + \sum_{i<=n} \{({}^nC_i * 2^{(i-2)}) - ({}^nC_i)\}$ ..... (4.2.5)

The algorithm proceeds as follows:

Generate initial solutions.

Repeat

Take first initial solution.

Repeat

Make this initial solution as current solution.

Find all feasible solutions in the neighbourhood of the current solution.

Evaluate all of them.

Find the best solution amongst them.

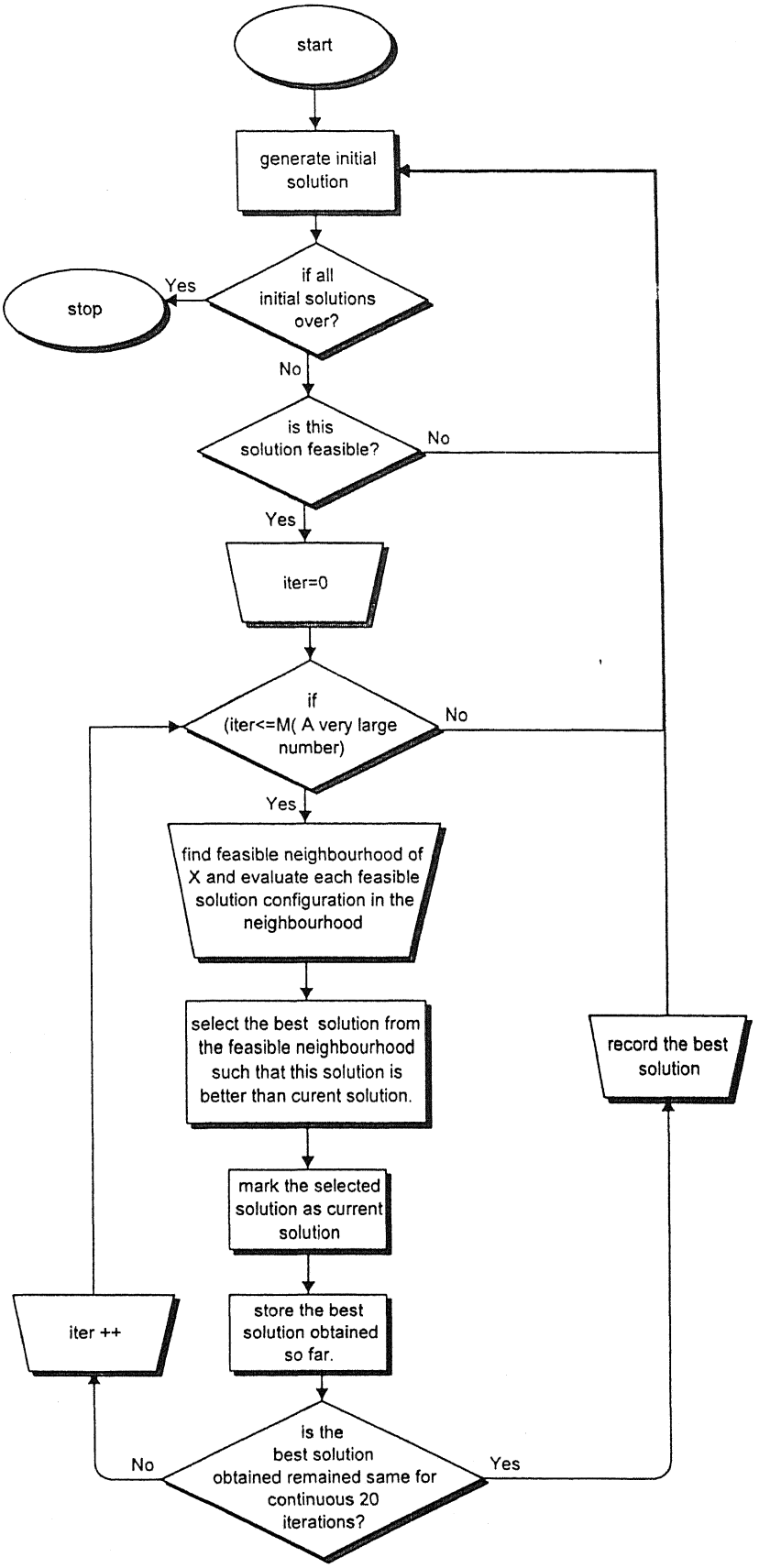
Make this as current solution.

Until (no improved solution is found or the best solution remains same for a number of iterations)

Until (all initial solutions are checked.)

Store the best value obtained.

Greedy Search Flow Chart



4.3 Implementation and results obtained:

We implemented this algorithm using JAVA. The code is given in Appendix. We tested the algorithm on various clash situations. Here is result:

Four satellite visibilities are clashing and their visibility start timings and visibility end timings are as shown follows-

Vis_start_sat1 = 6520	Vis_end_sat1 = 7115
Vis_start_sat2 = 7244	Vis_end_sat2 = 8318
Vis_start_sat3 = 7712	Vis_end_sat3 = 8650
Vis_start_sat4 = 8027	Vis_end_sat4 = 9196

Case A: *Linear objective function*

$F = \sum_{\text{over all } i}\{(e_i - s_i)*v_i\}$

Case B: *Non Linear objective function.*

$F = \sum_{\text{over all } i}\{A_i+(1-e^{-(ci-si)*.00958} * v_i)/.00958\}$

A_i= minimum support time * v_i

c_i = end of support of satellite i

s_i = start of support of satellite i

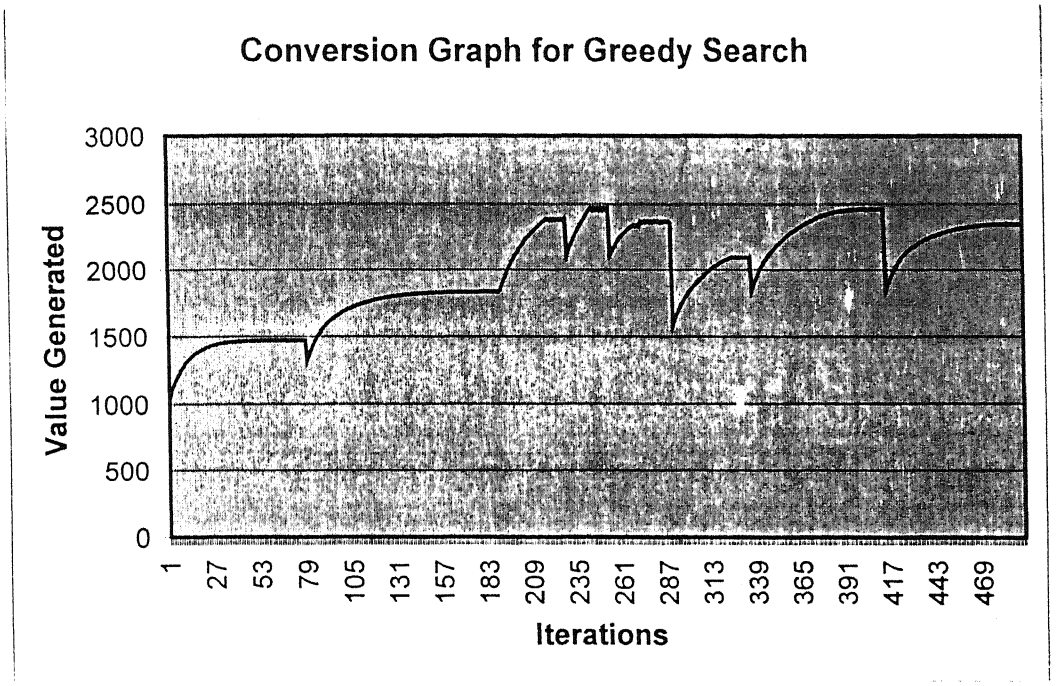
v_i = value contributed to F per unit time by supporting satellite i

Table 4.1 Results obtained using Greedy Search

Problem Scenario	Satellite Number, i	v_i	**Case A:** Linear objective function				**Case B:** Non-linear objective function.			
			s_i	e_i	$x_i = e_i - s_i$	Function Value	s_i	e_i	$x_i = e_i - s_i$	Function Value
1	1	1	6520	6520	0	4670	6520	6520	0	3415.99
	2	2	7244	7613	369		7244	7884	640	
	3	3	8213	8213	0		7712	7712	0	
	4	4	8213	9196	983		8484	9196	712	
2	1	4	6520	7115	595	4467	6520	7115	595	3855.55
	2	3	7715	8318	603		7715	8318	603	
	3	2	7712	7712	0		7712	7712	0	
	4	1	8918	9196	278		8027	8027	0	
3	1	1	6520	6520	0	4670	6520	6520	0	3415.99
	2	4	7244	8227	983		7244	7956	712	
	3	3	7712	7712	0		7712	7712	0	
	4	2	8827	9196	369		8556	9196	640	
4	1	1	6520	7112	592	4344	6520	7115	595	2881.89
	2	3	7712	7712	0		7712	7712	0	
	3	4	7712	8650	938		7715	8650	935	
	4	2	8027	8027	0		8027	8027	0	

4.4 Graphs:

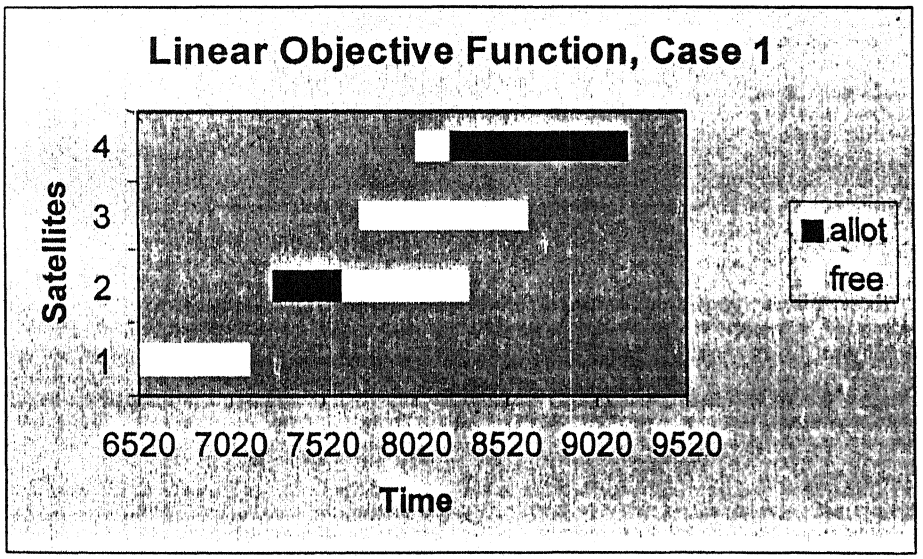
4.4.1 Conversion Graph of Greedy Algorithm



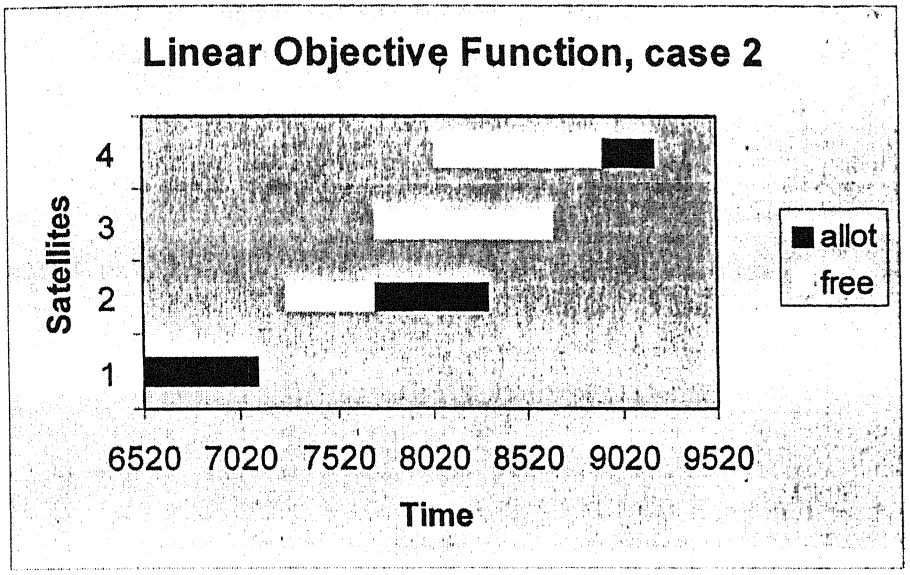
4.4.2 Satellite Support scenario Graphs

Linear Objective function:

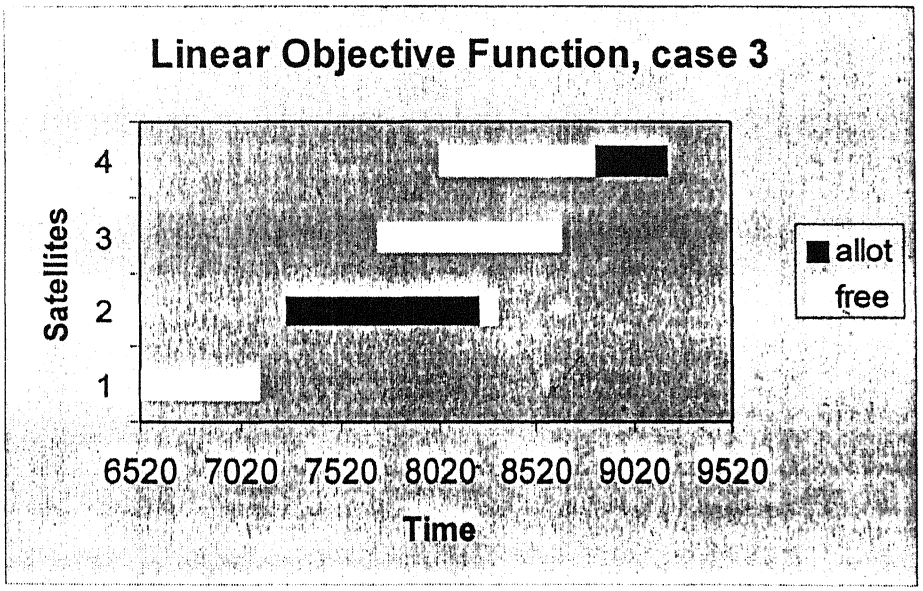
Problem case 1: v1=1 v2=2 v3=3 v4=4



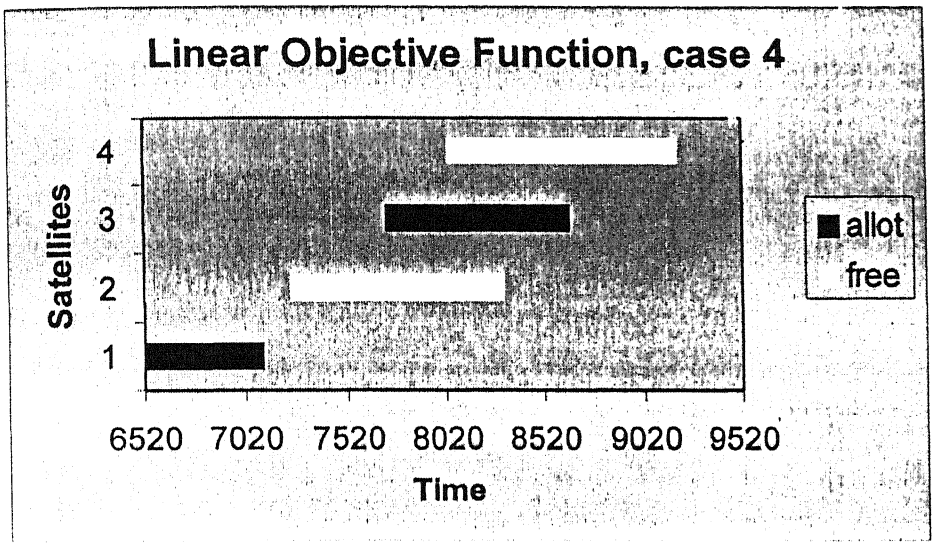
Problem case 2: v1 = 4 v2= 3 v3= 2 v4=1



Problem case 3: v1=1 v2=4 v3=3 v4=2

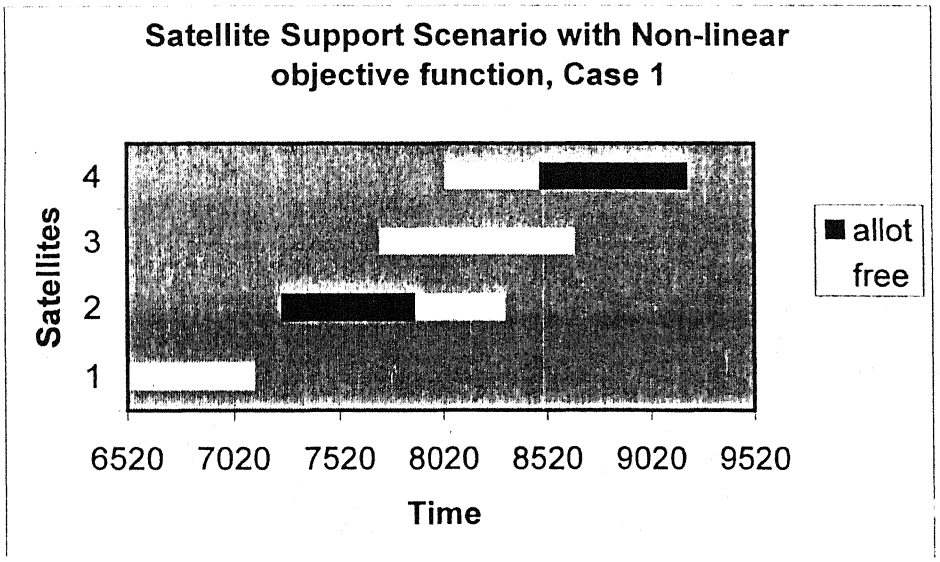


Problem case 4: v1=1 v2=3 v3=4 v4=2

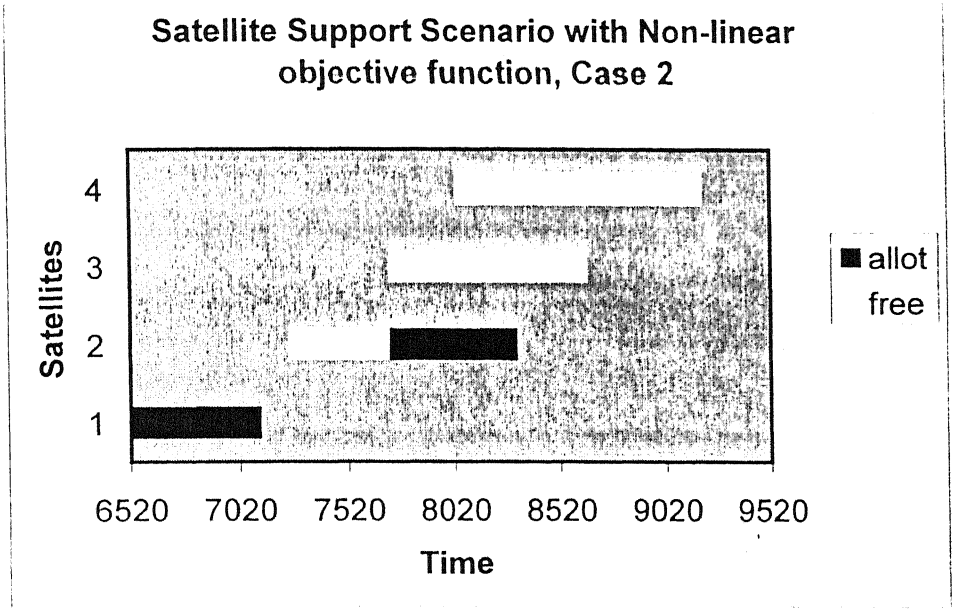


Non-Linear Objective Function.

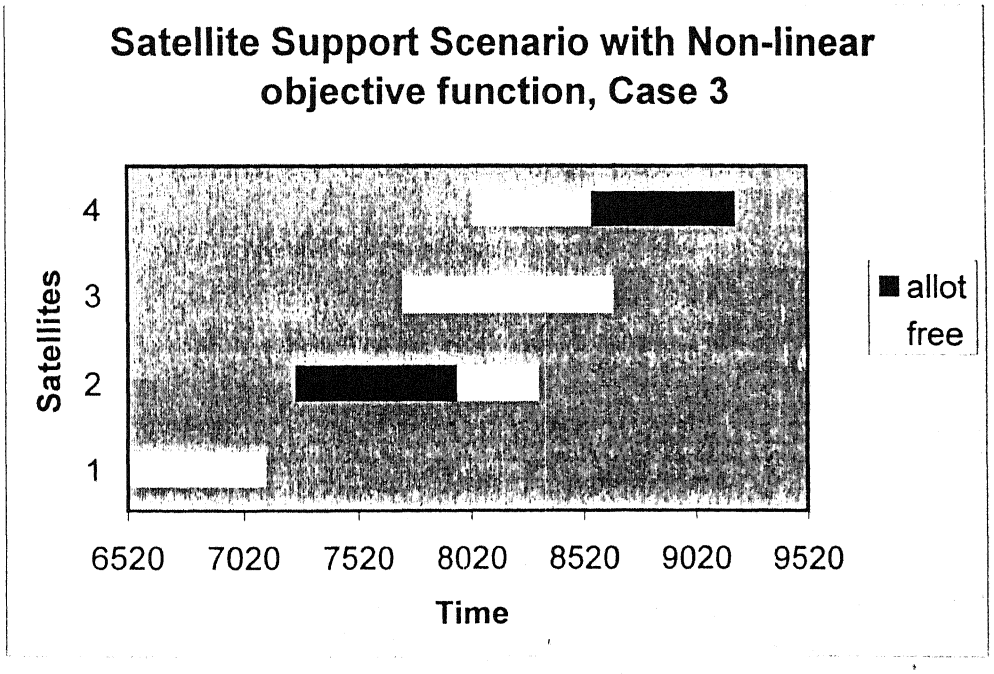
Problem case 1: v1 =1 v2=2 v3 = 3 v4=4



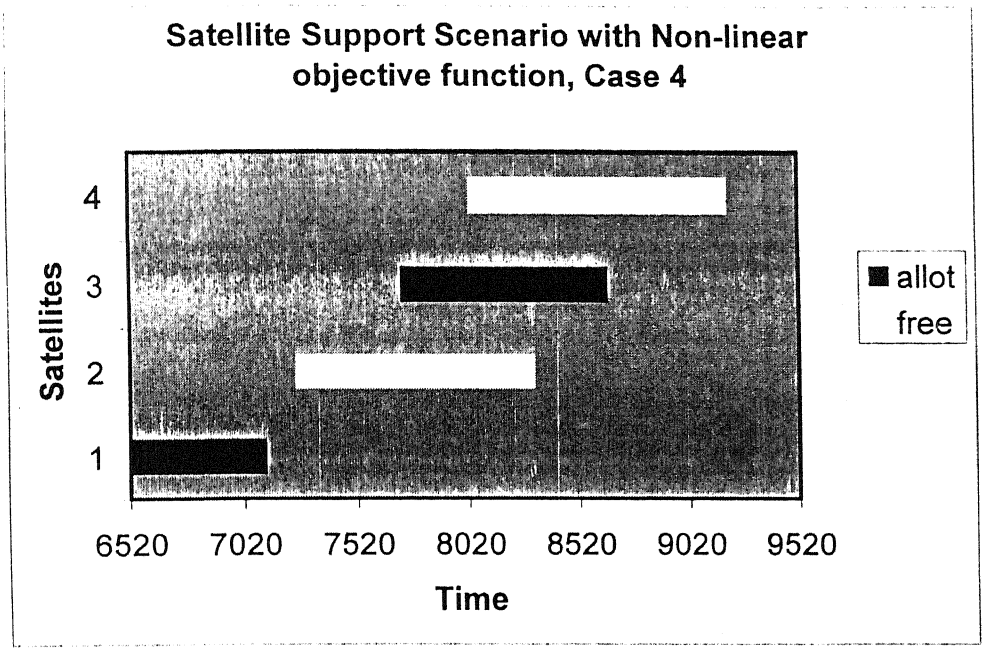
Problem case 2: v1=4 v2= 3 v3=2 v4=1



Problem case 3: v1=1 v2= 4 v3=3 v4=2



Problem case 4: v1=1 v2= 3 v3=4 v4=2



Chapter 5
Approach using Tabu Search

5.1 Tabu Search

Tabu search is an iterative procedure for solving discrete combinatorial optimisation problems. It was first suggested by Glover [7] and since then has become increasingly used. It has been successfully applied to obtain optimal or sub-optimal solutions to such problems as scheduling, timetabling, travelling salesperson and layout optimisation.

The basic idea of the method, described by Glover [7], is to explore the *search space* of all feasible solutions by a sequence of *moves*. A move from one solution to another is the best available. However, to escape from locally optimal but not globally optimal solutions and to prevent cycling, some moves, at one particular iteration, are classified as forbidden or *tabu* (or *taboo*). Tabu moves are based on the short-term and long-term history of the sequence of moves. A simple implementation, for example, might classify a move as tabu if the reverse move has been made recently or frequently. Sometimes, when it is deemed favourable, a tabu move can be overridden. Such *aspiration criteria* might include the case that, by forgetting that a move is tabu, leads to a solution which is the best obtained so far.

Formally, tabu search is applied to an optimisation problem as follows. Suppose F is the real-valued objective function on a search space S and it is required to find a $c \in S$ such that $F(c)$ has maximal value. For combinatorial hard (NP-complete) problems, this requirement needs to be relaxed to finding a $c \in S$ such that $F(c)$ is close to the maximal value (a *sub-optimal* value). This is because any known algorithm to determine the maximal solution requires time that is exponentially increasing with the problem size [17]. Sub-optimal problems may be solved by halting when a certain threshold for an acceptable solution has been achieved or when a certain number of iterations have been completed.

A characterisation of the search space S for which tabu search can be applied is that there is a set of k moves $M=\{m1,..mk\}$ and the application of the moves to a feasible

the variables changed during a move is taken as the attribute and used for enforcing tabu status.

The neighbourhood structure we are using is very simple. We add and subtract step size value to the current solution and get the neighbourhood solution. This we call as local neighbourhood. In our global neighbourhood we try to make a big jump in the search space through diversification. We use problem specific knowledge to define a number of diversified solutions.

We diversify our search to a new region when we find no improving solution in the current region for a number of moves. After jumping to the new search area we intensify our search in that area. The main advantage in this particular problem is that we already know which are the most promising areas and can limit our search to that only.

5.2.1 Results obtained

We tested our algorithm on various clash situations. One of them is as shown below. Four satellites are clashing and their visibility start timings and visibility end timings are as shown follows-

Vis_start_sat1 = 6520	Vis_end_sat1 = 7115
Vis_start_sat2 = 7244	Vis_end_sat2 = 8318
Vis_start_sat3 = 7712	Vis_end_sat3 = 8650
Vis_start_sat4 = 8027	Vis_end_sat4 = 9196

Case A: *Linear objective function*

$F = \sum_{\text{over all } i}\{(e_i-s_i)*v_i\}$

Case B: *Non Linear objective function.*

$F = \sum_{\text{over all } i}\{\Lambda_i+(1-e^{-(ci-si)*.00958} * v_i)/.00958\}$

Λ_i = minimum support time* $_i$

e_i = end of support of satellite i

s_i = start of support of satellite i

v_i = priority weight of satellite i

Flow Chart for Tabu Search

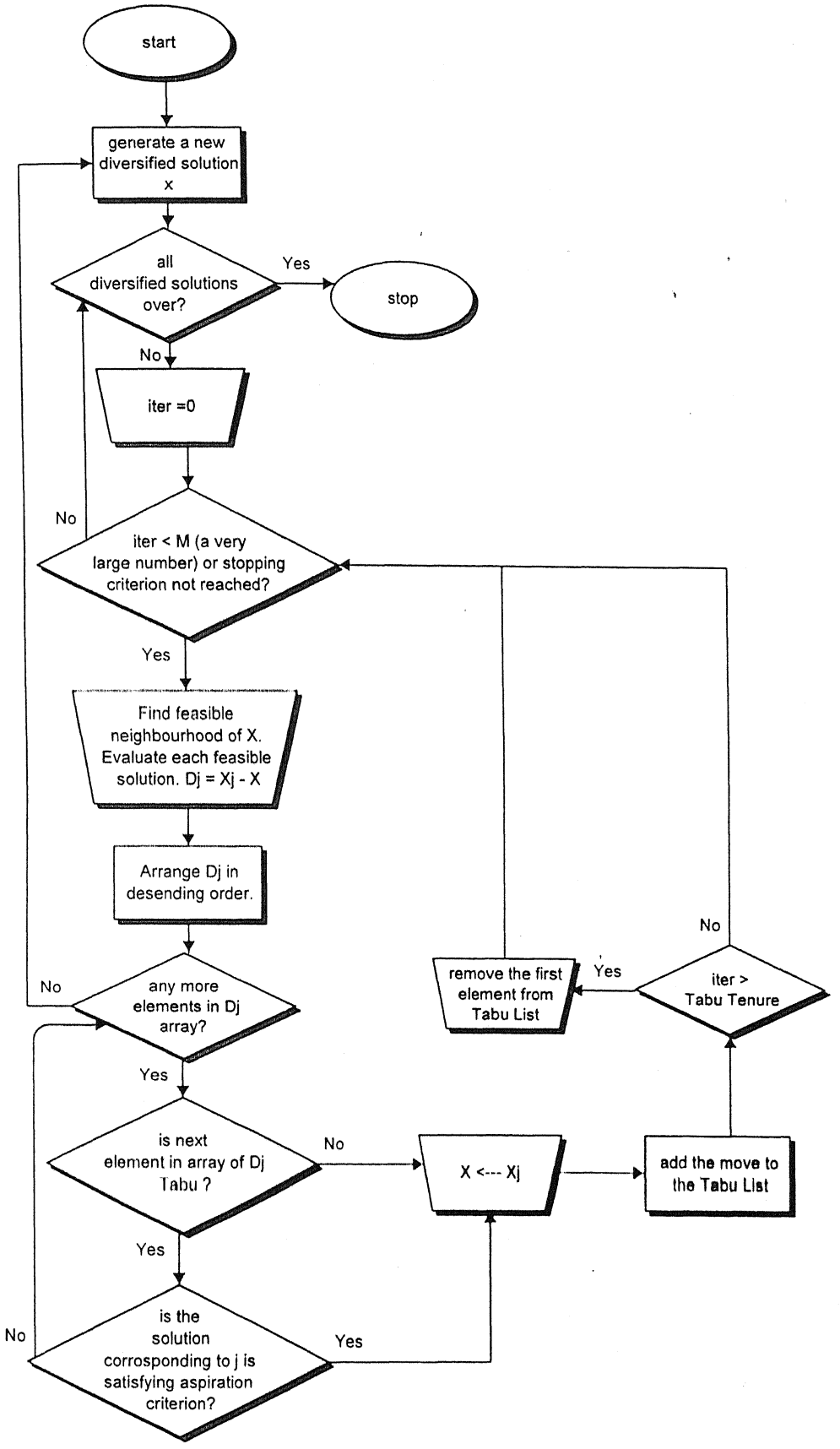
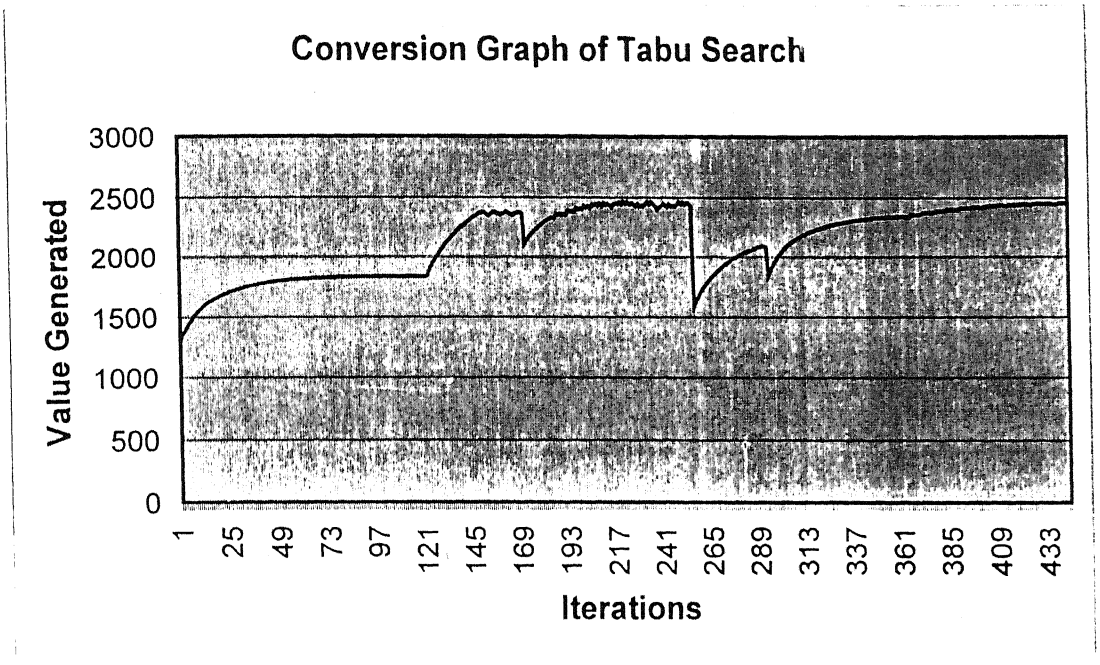


Table 5.1 Results obtained using Tabu Search:

Problem Scenario	Satellite Number, i	w_i	**Case A**: Linear objective function				**Case B**: Non-linear objective function.			
			s_i	e_i	$x_i = e_i - s_i$	Function Value	s_i	e_i	$x_i = e_i - s_i$	Function Value
1	1	1	6520	6520	0	4670	6520	7000	480	3481.79
	2	2	7244	7613	369		7600	8080	480	
	3	3	8213	8213	0		7712	7712	0	
	4	4	8213	9196	983		8680	9196	516	
2	1	4	6520	7115	595	4467	6520	7033	513	3962.04
	2	3	7715	8318	603		7633	8116	483	
	3	2	7712	7712	0		7712	7712	0	
	4	1	8918	9196	278		8716	9196	480	
3	1	1	6520	6520	0	4670	6520	7000	480	3481.79
	2	4	7244	8227	983		7600	8116	516	
	3	3	7712	7712	0		7712	7712	0	
	4	2	8827	9196	369		8716	9196	480	
4	1	1	6520	7112	592	4344	6520	7000	480	2971.34
	2	3	7712	7712	0		7600	8116	516	
	3	4	7712	8650	938		7712	7712	0	
	4	2	8027	8027	0		8716	9196	480	

5.3 Graphs

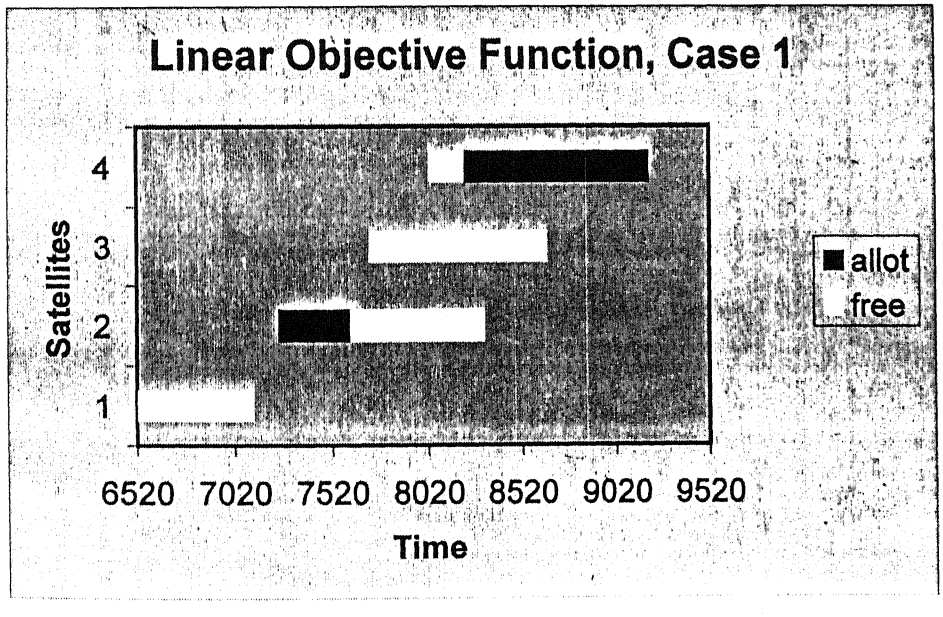
5.3.1 Conversion Graph for Tabu Search:



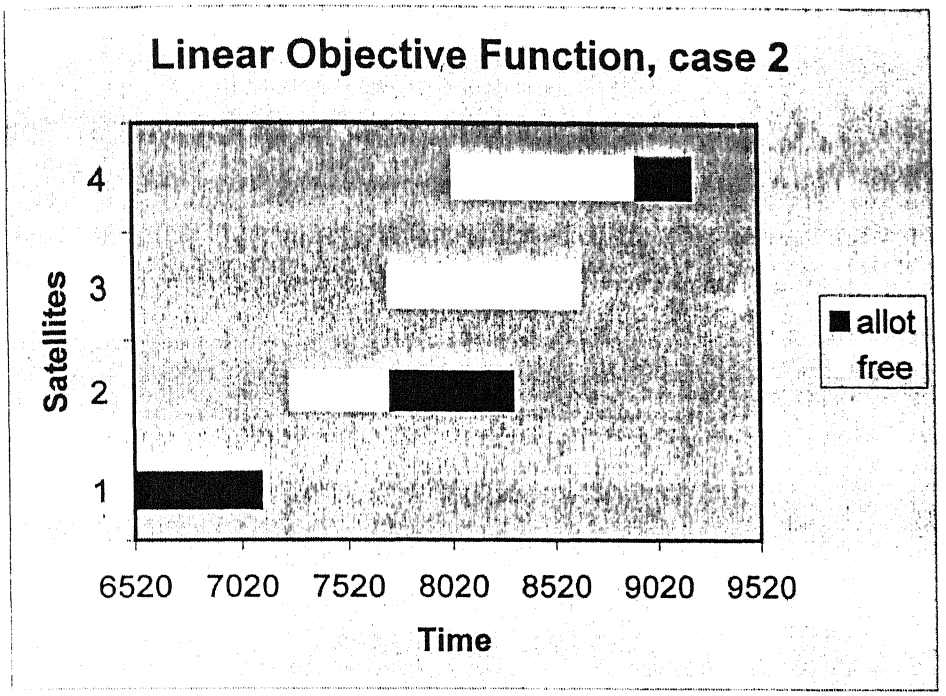
5.3.2 Satellite Support Scenario Graphs:

Linear Objective function:

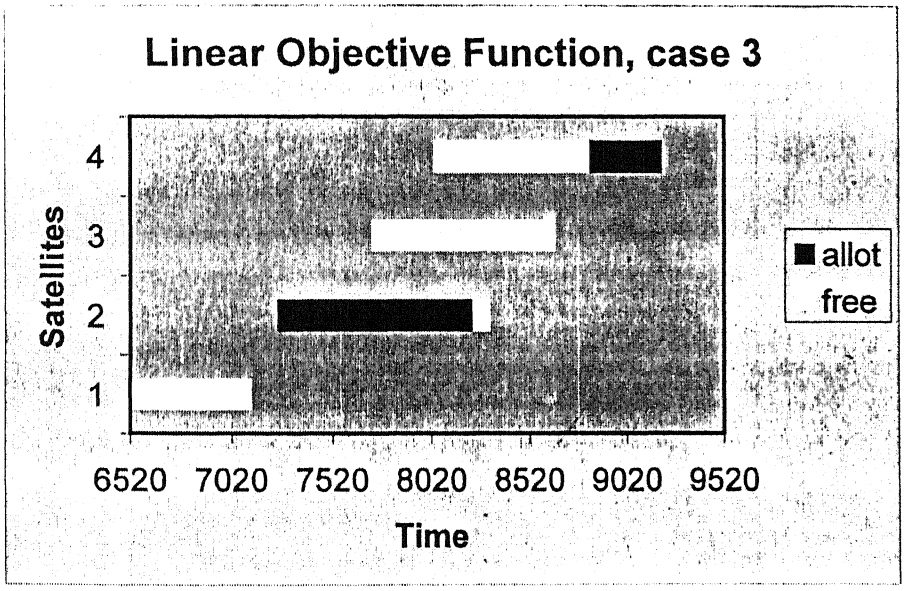
Problem case 1: v1=1 v2=2 v3=3 v4=4



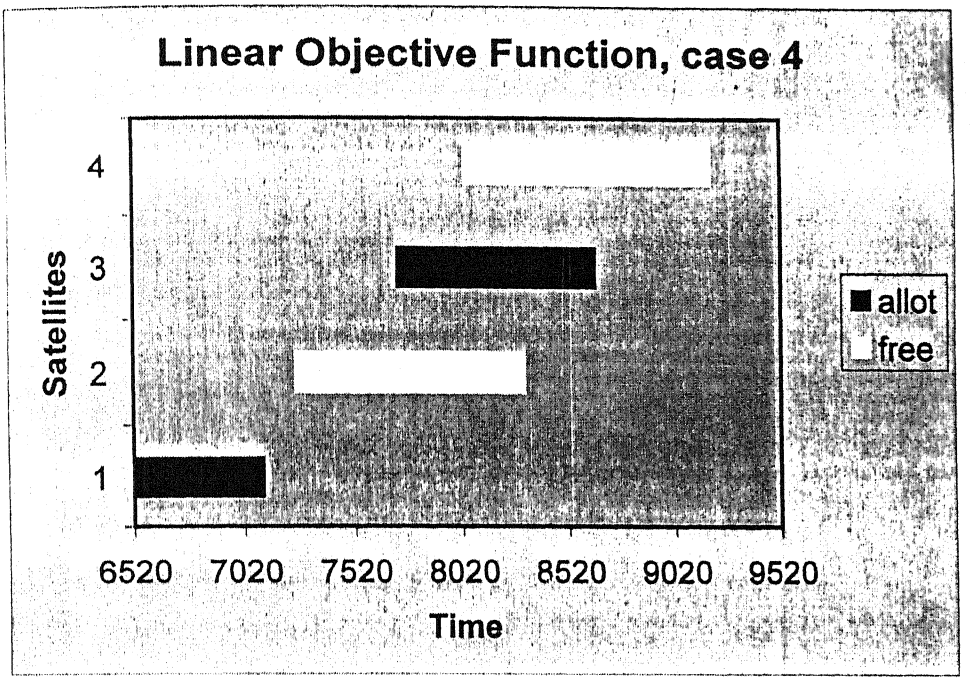
Problem case 2: v1 = 4 v2= 3 v3= 2 v4=1



Problem case 3: v1=1 v2=4 v3=3 v4=2

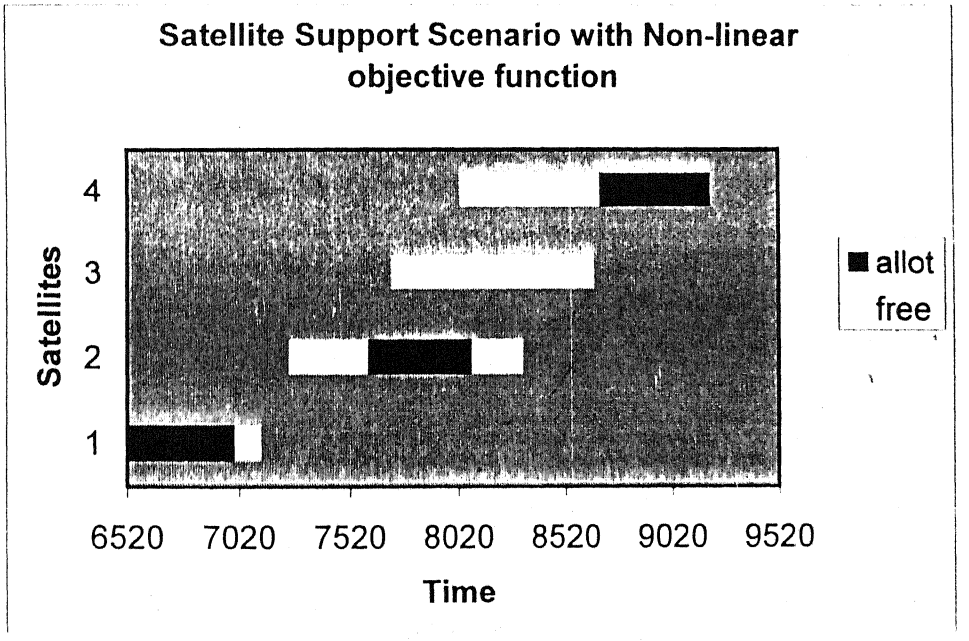


Problem case 4: v1=1 v2=3 v3=4 v4=2

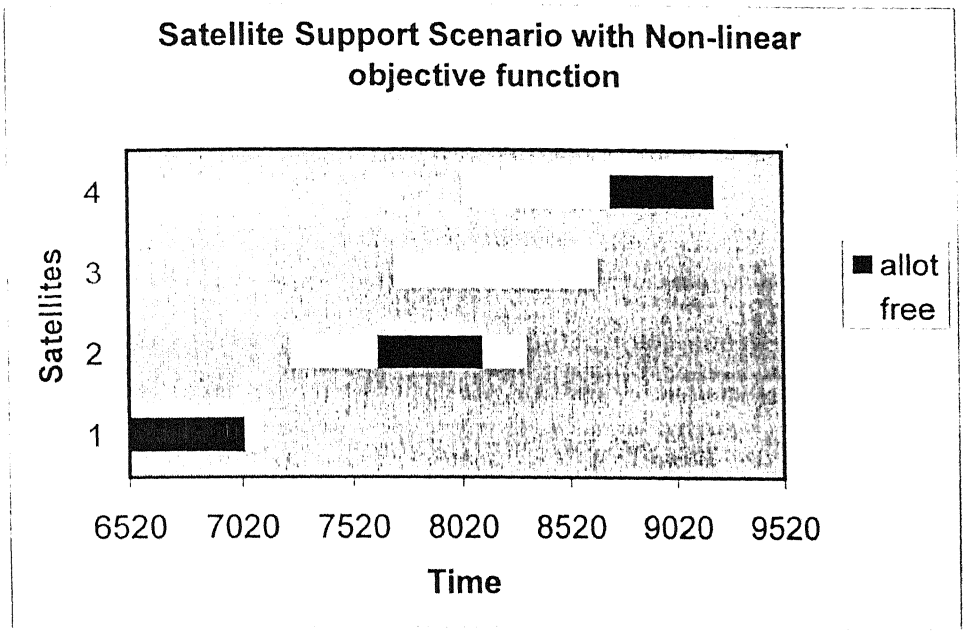


Non Linear Objective Function:

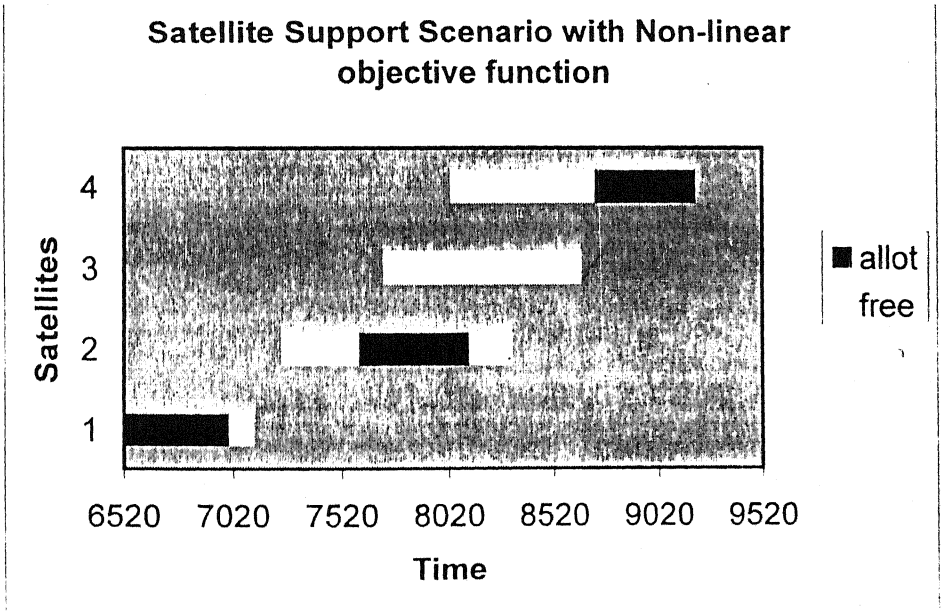
Problem case 1: v1=1 v2=2 v3=3 v4=4



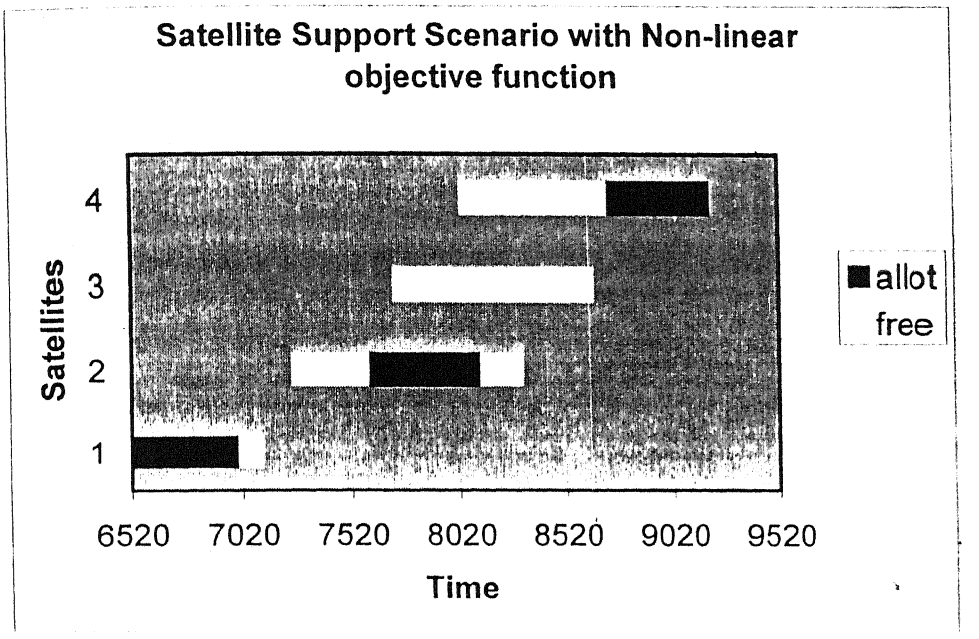
Problem case 2: v1=4 v2=3 v3=2 v4=1



Problem case 3: v1=1 v2=4 v3=3 v4=2



Problem case 4: v1=1 v2=3 v3=3 v4=2



Chapter 6

Approach using Simulated Annealing

6.1 Simulated Annealing:

Simulated annealing is a numerical optimisation technique based on the principles of thermodynamics. Annealing refers to the process in which a solid material is first melted and then allowed to cool by slowly reducing the temperature. The particles of the material attempt to arrange themselves in a low energy state during the cooling process. The collective energy states of the ensemble of particles can be considered the "configuration" of the material. The probability that a particle is at any energy level can be calculated by use of the Boltzmann distribution. As the temperature of the material decreases, the Boltzmann distribution tends toward the particle configuration that has the lowest energy. Metropolis et al [11] first realized that the thermal equilibrium process could be simulated for a fixed temperature by Monte Carlo methods to generate sequences of energy states [11]. The system is perturbed to yield a new configuration of the particles. The energy level before perturbation (Es) and the energy level after perturbation (Ei) are compared. If Es is greater than Ei (i.e., $E > 0$), the new perturbed system is accepted as the new configuration of particles. If $E > 0$, the probability of accepting the perturbed system follows the Metropolis criterion shown in following equation

$$p = \exp(-E/kT)$$

where k is the Boltzmann constant and T is a fixed temperature. Using this criterion, the material will eventually reach its equilibrium configuration.

This basic concept can be applied to numerical optimisation problems. Simulated annealing (SA) applies the Metropolis criterion to a series of variable settings (configurations) for the system being optimised. The new variable settings are obtained by perturbing the current configuration and can be thought of as steps or movements on the response surface. For numerical optimisation, the response (objective) function score (R) replaces the energy terms. The concept of temperature is retained. However, it is now combined with k and used as an important control

factor. The probability of accepting a detrimental step (i.e., R > 0, assuming minimization) is governed by following equation

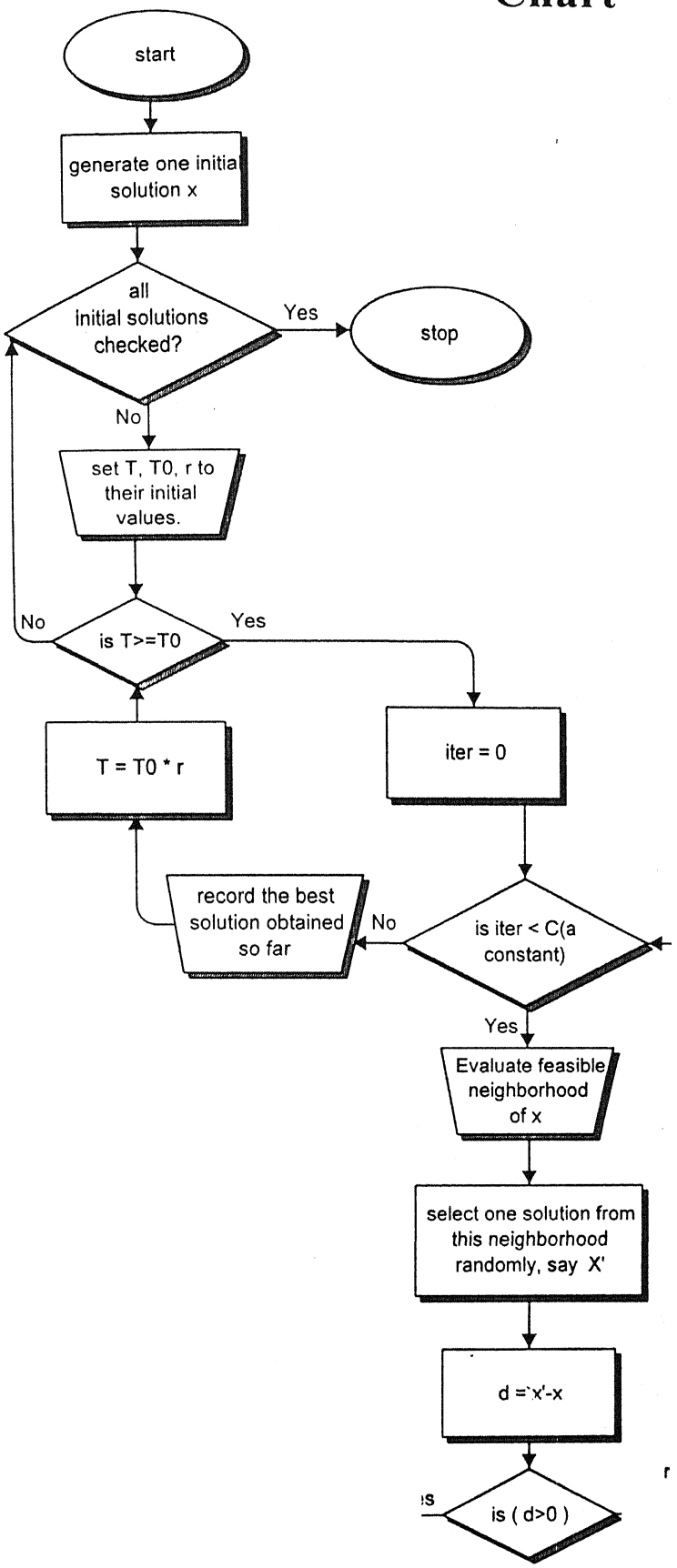
$$p = \exp(-R/T)$$

A random number, P, is drawn from a uniform random distribution on the interval [0,1]. If P > p, the detrimental step is rejected and a new step taken from the current position. If p< P, the detrimental step is accepted and the new configuration replaces the old one. A new step is then taken relative to this configuration. This criterion allows the possibility of a new variable being accepted as the new configuration even when it has a worse response function score than the current configuration. Moves that are very poor (i.e., large R) are less likely to be accepted than moves that are not poor. This feature allows the algorithm to "walk" from local optima. New steps are taken until some termination criterion is reached.

Analogous to the physical process, T is slowly reduced causing the probability of accepting a poor move to decrease with time. The schedule by which T is reduced is called the cooling schedule and is critical to the success of SA. Two important parameters governing the cooling schedule are the step size for perturbation and T. The parameter settings suggested by most researchers are step sizes and T values that allow approximately 80% of the poor moves to be accepted [12].

The primary advantage of SA is the ability to move from local optima. Thus, the ability to find the global optimum is not related to the initial conditions (i.e., the starting point). SA is also very simple to implement. The primary disadvantages to SA are the subjective nature of choosing the SA configuration parameters (e.g., T and step size) and that it typically requires more response or objective function evaluations than other optimisation approaches. SA has been termed a "biased random walk". Unlike other optimisation approaches that attempt to make intelligent moves on the response surface, the steps in SA are taken randomly. Thus, SA is classified as having a weak search heuristic. The fact that SA employs no knowledge of the response surface can be an advantage or a disadvantage depending on the application.

Chart


start
generate one initial solution x
all initial solutions checked?
Yes
stop
No
set T, T0, r to their initial values.
No
is T>=T0
Yes
T = T0 * r
iter = 0
record the best solution obtained so far
No
is iter < C(a constant)
Yes
Evaluate feasible neighborhood of x
select one solution from this neighborhood randomly, say X'
d = 'x'-x
is (d>0)

6.2 SA on this problem

The basic algorithm remains same. As we have do deal with discontinuous nature of search space, we have to have some mechanism to make a jump in the search space. One way is to define the neighbourhood structure in such a way that the selected solution from neighbourhood automatically jumps to the unexplored region. This makes the number of neighbourhood solutions very large that is of the order of (2^n raise to 2^nraise to 2^n...upto (n+1)), which is very difficult to construct and not practical.

In our case the problem structure is such that we can jump in the search space if we define some initial solutions already; its equivalent to starting algorithm with different initial solutions in greedy search. And here the advantage is that we can have only 2^n different solutions at hand from where we can start.

As mentioned earlier, in this problem, the individual hills in the search space have different peaks, but SA can skip away from local optima and find the global optima in individual hill.

6.3 Implementation and Parameter optimisation.

The critical parameters in simulated annealing are, the initial high temperature and the fraction by which initial temperature is brought down to freezing point. This is called cooling schedule. We tried to find out best value of T (Initial Temperature) and α (fraction by which T is reduced).

Results:

We considered the following sample problem:

Vis_start_sat1 = 6520	Vis_end_sat1 = 7115
Vis_start_sat2 = 7244	Vis_end_sat2 = 8318
Vis_start_sat3 = 7712	Vis_end_sat3 = 8650
Vis_start_sat4 = 8027	Vis_end_sat4 = 9196

Case A: Linear objective function

$F = \sum_{\text{over all } i}\{(c_i - s_i)*v_i\}$

Case B: Non Linear objective function.

$F = \sum_{\text{over all } i}\{A_i + (1-e^{-(ci-si)*.00958} * v_i)/.00958\}$

where,

Λ_i= initial support time * w_i

e_i = end of support of satellite i

s_i = start of support of satellite i

v_i = priority weight of satellite i

Table shows the average and standard deviation of the best value obtained after five different seeds given to random number generator.

Table 6.1 Average of the Best function value Generated for different values of T & α using different seeds for random number generator.

1. Linear objective function.

α ↓	T →	10	15	20	25	30
0.5	Avg.	4344	3602	4344	4344	4344
	SD	0	0	0	0	0
0.7	Avg.	4344	4344	4344	4344	4344
	SD	0	0	0	0	0
0.9	Avg.	4344	4344	4344	4344	4344
	SD	0	0	0	0	0
0.99	Avg.	4344	4344	4344	4344	4344
	SD	0	0	0	0	0

2. Non-linear objective function.

α ↓	T →	10	15	20	25	30
0.5	Avg.	2762.56	2764.77	2764.77	2762.56	2762.56
	SD	3.7610	4.633	4.633	4.805	5.085
0.7	Avg.	2764.77	2769.89	2764.77	2769.89	2769.89
	SD	4.633	4.536	4.633	4.536	4.536
0.9	Avg.	2769.89	2769.89	2769.89	2971.34	2971.34
	SD	4.536	2.609	2.609	0	0
0.99	Avg.	2769.89	2971.34	2769.89	2971.34	2971.34
	SD	2.609	0	2.609	0	0

Though there is no effect of seed on best value obtained when using linear objective function, above table (with non-linear objective function values) clearly indicate that higher the initial temperature and finer the fraction better is the result. Hence we select the initial temperature value T as 25 and fraction $\alpha = .99$.

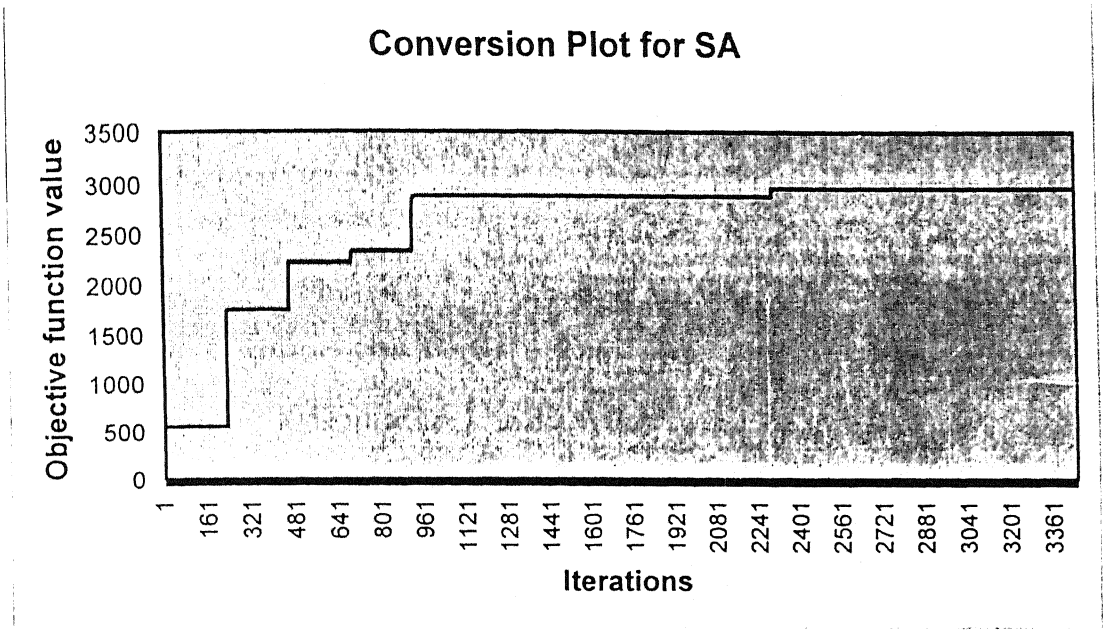
Following Table shows the best objective function value obtained by setting T=25 and $\alpha = .99$.

Table 6.2 Results obtained using Simulated Annealing

Problem Scenario	Satellite Number, i	w_i	**Case A**: Linear objective function				**Case B**: Non-linear objective function.			
			s_i	e_i	$x_i = e_i - s_i$	Function Value	s_i	e_i	$x_i = e_i - s_i$	Function Value
1	1	1	6520	6520	0	4670	6520	7000	480	3481.79
	2	2	7244	7613	369		7600	8080	480	
	3	3	8213	8213	0		7712	7712	0	
	4	4	8213	9196	983		8680	9196	516	
2	1	4	6520	7115	595	4467	6520	7033	513	3962.04
	2	3	7715	8318	603		7633	8116	483	
	3	2	7712	7712	0		7712	7712	0	
	4	1	8918	9196	278		8716	9196	480	
3	1	1	6520	6520	0	4670	6520	7000	480	3481.79
	2	4	7244	8227	983		7600	8116	516	
	3	3	7712	7712	0		7712	7712	0	
	4	2	8827	9196	369		8716	9196	480	
4	1	1	6520	7112	592	4344	6520	7000	480	2971.34
	2	3	7712	7712	0		7600	8116	516	
	3	4	7712	8650	938		7712	7712	0	
	4	2	8027	8027	0		8716	9196	480	

6.4 The Graphs:

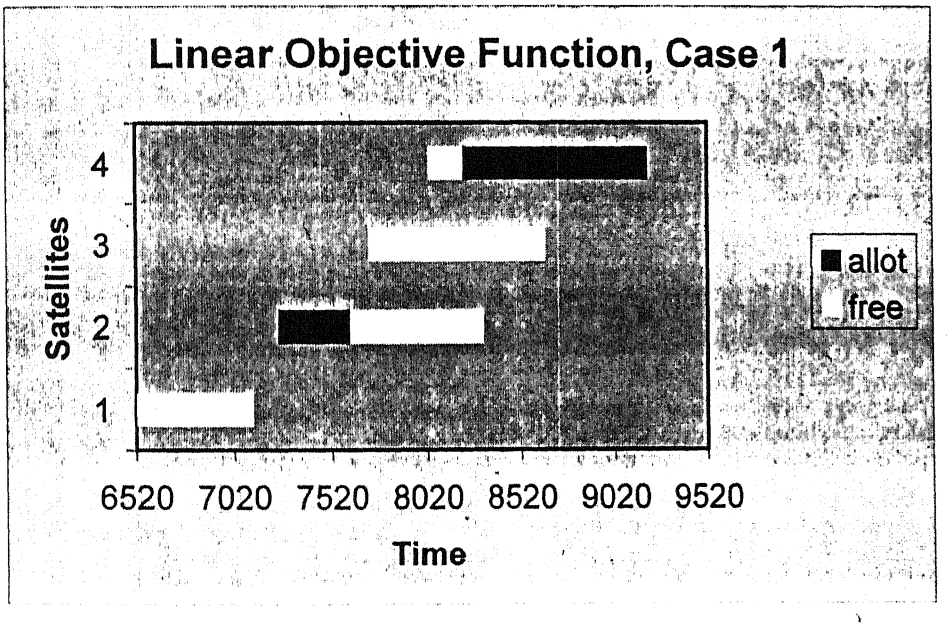
6.4.1 Conversion Graph for SA.



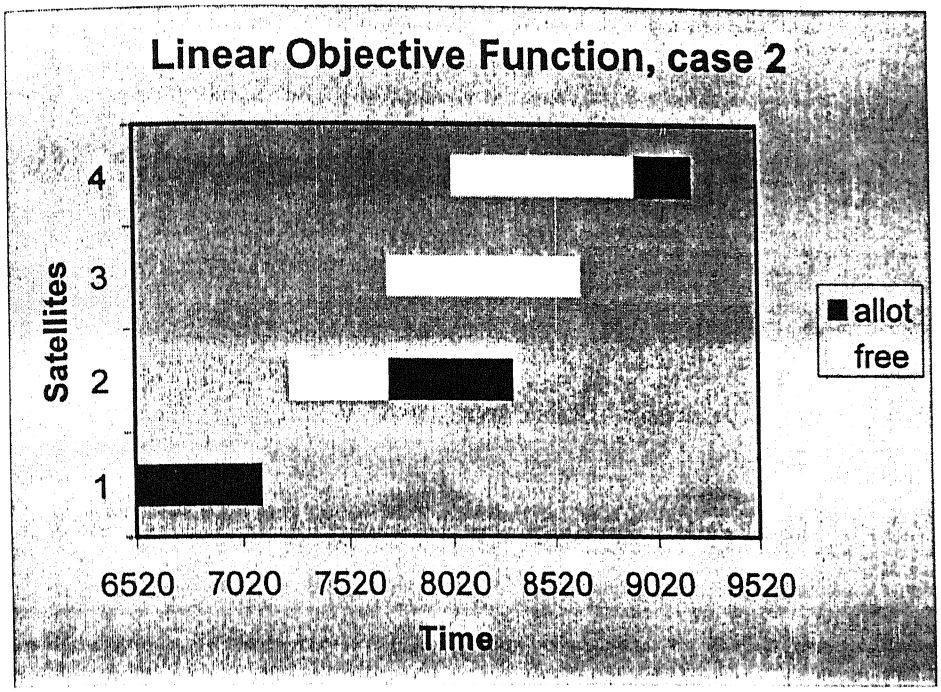
6.4.2 Satellite Support Scenario graphs

Linear Objective function:

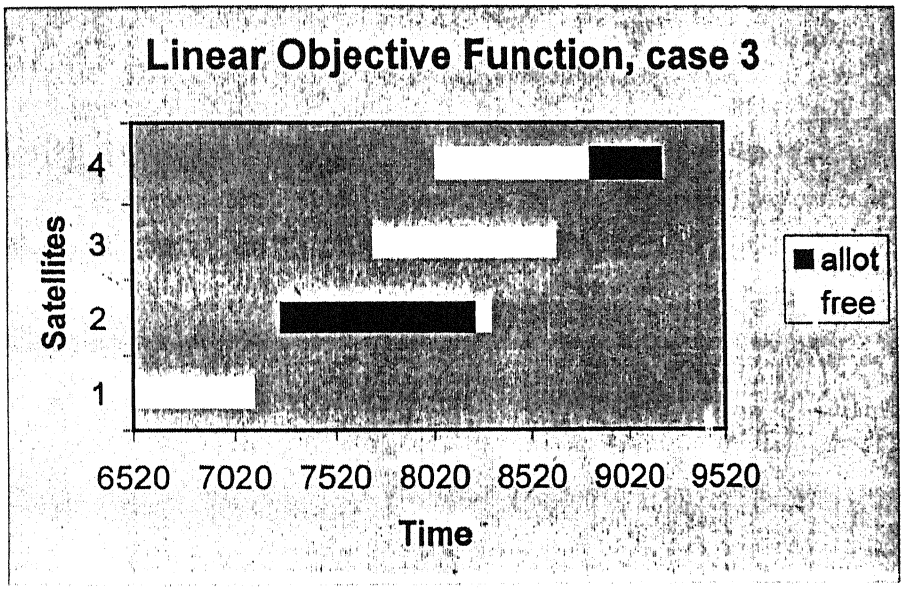
Problem case 1: v1=1 v2=2 v3=3 v4=4



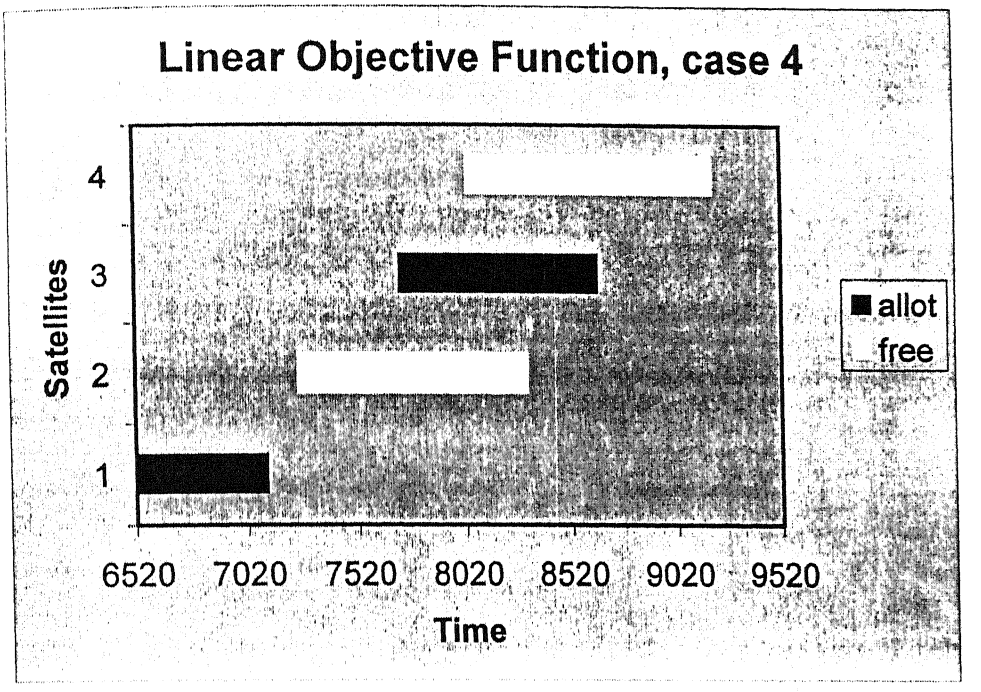
Problem case 2: v1 = 4 v2= 3 v3= 2 v4=1



Problem case 3: v1=1 v2=4 v3=3 v4=2

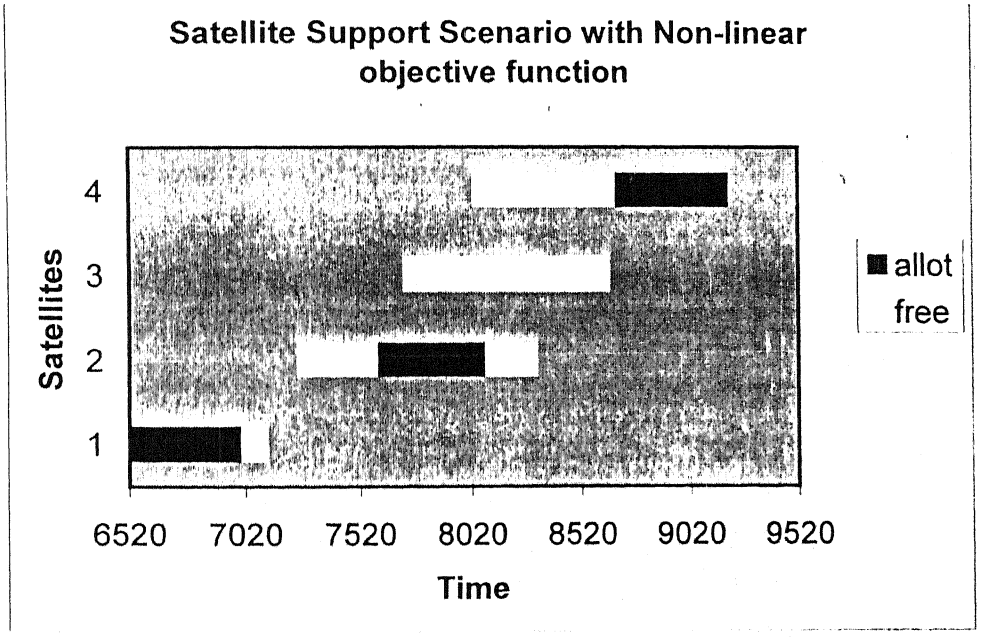


Problem case 4: v1=1 v2=3 v3=4 v4=2

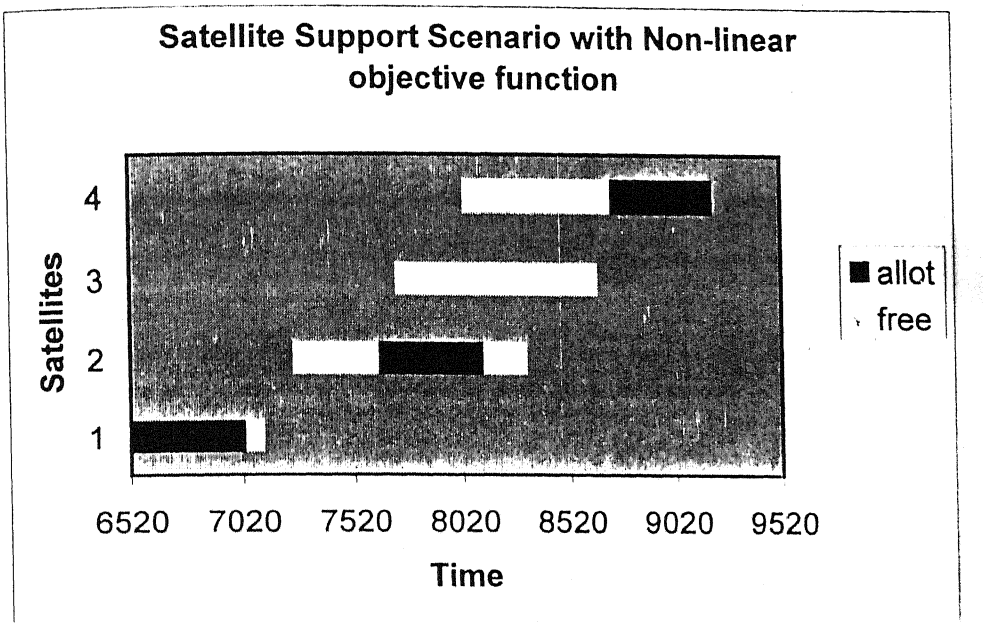


Non Linear Objective Function:

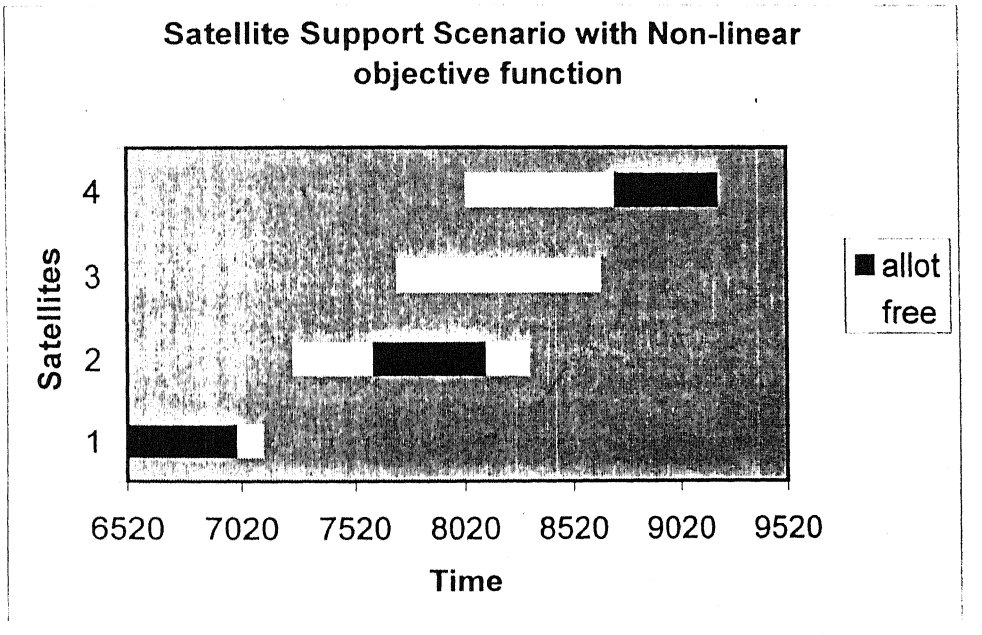
Problem case 1: v1=1 v2=2 v3=3 v4=4



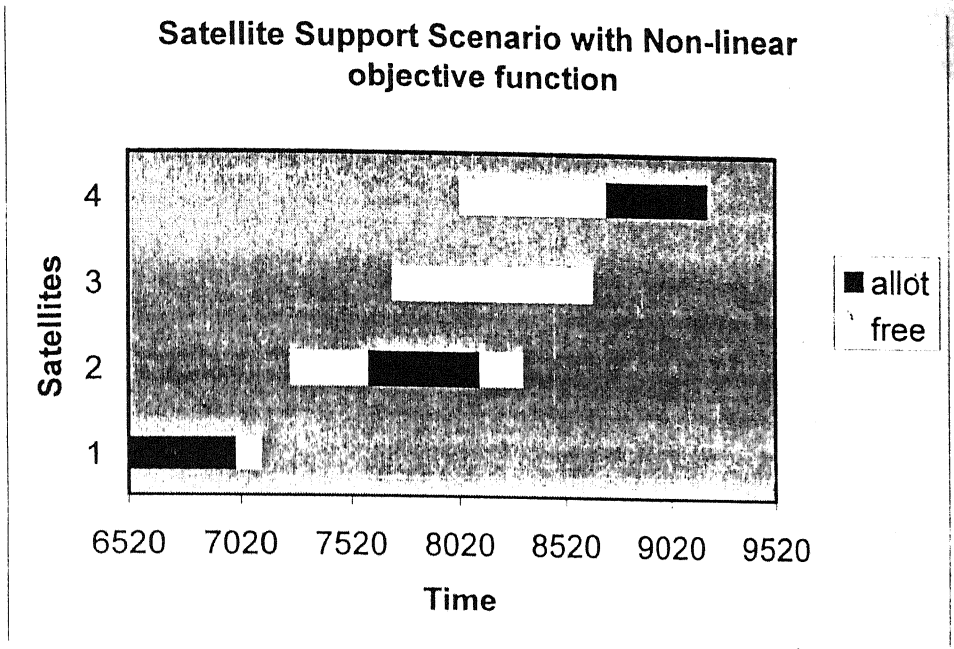
Problem case 2: v1=4 v2=3 v3=2 v4=1



Problem case 3: v1=1 v2=4 v3=3 v4=2



Problem case 4: v1=1 v2=3 v3=3 v4=2



Chapter 7
Approach using Genetic Algorithm

7.1 Genetic Algorithm

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomised information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. While randomised, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance. [8]

7.2 GA on this Problem

Kumar and Bagchi (2000) solved this visibility clash resolution problem using GA. We describe here the approach given by them.

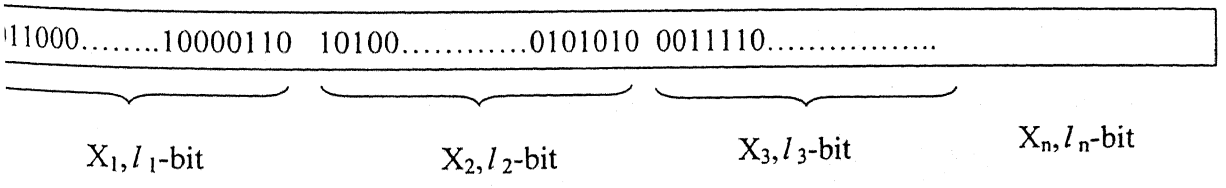
7.2.1 GA Chromosome Construction

They use binary coding to code the chromosome because in this problem the decision variables $\{x_i\}$ are discrete numerical quantities. A single chromosome would contain all information for the full duration of support for each satellite involved in a clashed set of visibilities at a station.

For example:

At a station 4 satellite visibilities are clashing, then the chromosome would contain values for X_1, X_2, X_3, and X_4.

A typical chromosome would look as below.



7.2.2 Deciding the Length of the Chromosome

In GA formulation using binary coded variables, the length of a substring representing real a variable depends on desired accuracy of that variable. If one do ℓ_i-bit coding for a variable, the obtainable accuracy in that variable is approximately

$$\left(x_i^{(U)} - x_i^{(L)}\right)/2^{\ell_i}$$

In ISRO's case the upper limit on operation timing is 16 Minutes 23 Second. Thus the attainable accuracy with a 10- bit substring for all x_i's will be:

$$(16*60+23)/2^{10}=0.959 \text{ seconds}$$

this level of precision will always keep the error to be less then one second if one selects ℓ_i =10 bits.

7.2.3 Feasibility Assurance

The feasibility of a species is maintained by incorporating constraints inside chromosome coding. This is done by taking modified upper and lower limits of possible values of time.

For example, consider decision variable x_i .

The limits for x_1 are $[a_1, b_1]$ (Fig 3.2)

And limits for X_2 will be from

[(Greater of e_1+r and a_2) and b_2]

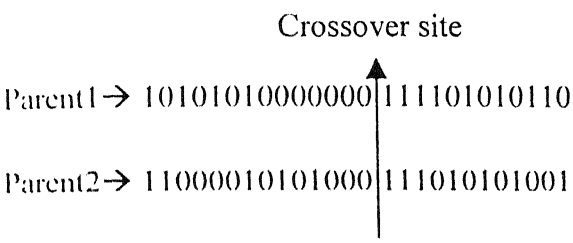
the above approach would ensure feasibility of all chromosomes.

7.2.4 GA Operators

7.2.4.1 Crossover

In the crossover operation, new strings are created by exchanging information among strings of the mating pool . In this application they used single point crossover.

The Crossover used here is one-cut point and it exchanges the right part of two parents to generate offspring . The single point crossover operator is performed by randomly choosing a crossing site along the string and by exchanging all bits on right side of crossing site between the two parents . As applied to this problem, a typical crossover is shown as below.



Child1→ 10101010000000111010101001

Child2→ 11000010101000111101010110

7.2.4.2 Mutation

Mutation alters one or more genes with a probability equal to the mutation rate . So, if the probability of mutation is set at 0.01, so we expect that, on average, 1% of total bit of the population of solution would undergo mutation.

7.2.5 Fitness Function

For calculation of fitness function (Value function), first the decoded values of chromosome string are calculated by a linear mapping rule. The rule used is:

$$x_i = x_i^{(L)} + \frac{x_i^{(U)} - x_i^{(L)}}{2^{r_i} - 1} decodedvalue(s_i)$$

Here decoded value $\{x_i\}$ is the integer conversion of binary string $\{s_i\}$.

The decoded value of binary string is calculated as:

Decoded value $(s_i) = \sum_{i=0}^{l-1} 2^i s_i$ Where $s_i \in (0,1)$

After all x_i's are calculated, total value function of the chromosome may be calculated by Eq.3.4.1 as:

$P = \sum_{\text{over all } i}\{v_i * x_i\}$

Or by Eq. 3.3.5 as:

$P = \sum_{\text{over all } i}\{A_i + (1 - e^{-(ei-si)*.00958} * v_i)/.00958\}$

7.2.6 Selection

Selection strategy used in the their work is Elitist selection. Elitist selection ensures that the best chromosome is passed into the new generation. In this strategy all individual fitness values are scanned and best member is selected. Number of selections made is such that the size of population remains constant.

7.3 Implementation and Results

They tested the algorithm on a real life situation problem of clashing visibilities. The visibilities are shown below.

Vis_start_sat1 = 6520	Vis_end_sat1 = 7115
Vis_start_sat2 = 7244	Vis_end_sat2 = 8318
Vis_start_sat3 = 7712	Vis_end_sat3 = 8650
Vis_start_sat4 = 8027	Vis_end_sat4 = 9196

Case A: *Linear objective function*

Maximize,

$F = \sum_{\text{over all } i}\{(e_i - s_i) * v_i\}$

Case B: *Non Linear objective function.*

Maximize,

$F = \sum_{\text{over all } i}\{A_i + (1 - e^{-(ei-si)*.00958} * v_i)/.00958\}$

A_i= minimum support time * v_i

e_i = end of support of satellite i

$$\text{Decoded value } (s_i) = \sum_{i=0}^{l-1} 2^i s_i \quad \text{Where } s_i \in (0,1)$$

After all x_i's are calculated, total value function of the chromosome may be calculated by Eq.3.4.1 as:

$P = \sum_{\text{over all } i}\{v_i * x_i\}$

Or by Eq. 3.3.5 as:

$P = \sum_{\text{over all } i}\{A_i + (1 - e^{-(ei-si)*.00958} * v_i)/.00958\}$

7.2.6 Selection

Selection strategy used in the their work is Elitist selection. Elitist selection ensures that the best chromosome is passed into the new generation. In this strategy all individual fitness values are scanned and best member is selected. Number of selections made is such that the size of population remains constant.

7.3 Implementation and Results

They tested the algorithm on a real life situation problem of clashing visibilities. The visibilities are shown below.

Vis_start_sat1 = 6520	Vis_end_sat1 = 7115
Vis_start_sat2 = 7244	Vis_end_sat2 = 8318
Vis_start_sat3 = 7712	Vis_end_sat3 = 8650
Vis_start_sat4 = 8027	Vis_end_sat4 = 9196

Case A: *Linear objective function*

Maximize,

$F = \sum_{\text{over all } i}\{(e_i - s_i) * v_i\}$

Case B: *Non Linear objective function.*

Maximize,

$F = \sum_{\text{over all } i}\{A_i + (1 - e^{-(ei-si)*.00958} * v_i)/.00958\}$

A_i= minimum support time * v_i

e_i = end of support of satellite i

s_i = start of support of satellite i

v_i = value contributed to F per unit time by supporting satellite i

Table 7.1

Problem Scenario	Satellite Number, i	w_i	**Case A**: Linear objective function				**Case B**: Non-linear objective function.			
			s_i	e_i	$x_i = e_i - s_i$	Function Value	s_i	e_i	$x_i = e_i - s_i$	Function Value
1	1	1	6520	6520	0	4670	6520	7000.01	483.4	3478.67
	2	2	7244	7613	369		7600.01	8082.5	480.4	
	3	3	8213	8213	0		7712	7712	0	
	4	4	8213	9196	983		8682.5	9196	512.2	
2	1	4	6520	7115	595	4467	6520	7023.55	503.5	3958.48
	2	3	7715	8318	603		7623.55	8115.62	492.1	
	3	2	7712	7712	0		7712	7712	0	
	4	1	8918	9196	278		8715.62	9196	480.3	
3	1	1	6520	6520	0	4670	6520	7000.1	480.1	3481.01
	2	4	7244	8227	983		7600.1	8115.8	515.7	
	3	3	7712	7712	0		7712	7712	0	
	4	2	8827	9196	369		8715.8	9196	480.1	
4	1	1	6520	7112	592	4344	6520	7000.07	480.07	2971.01
	2	3	7712	7712	0		7600.07	8115.01	516	
	3	4	7712	8650	938		7712	7712	0	
	4	2	8027	8027	0		8715.01	9196	480	

The parameters used for GA in above case are:

Pm= Probability of Mutation = 0.1

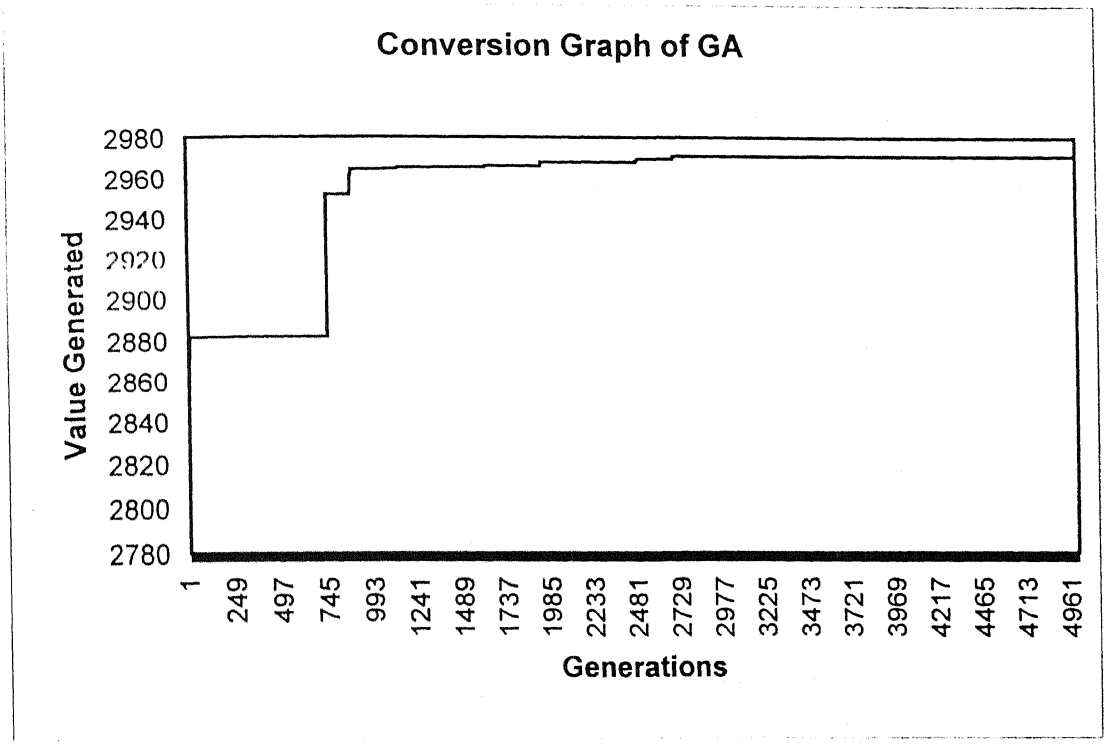
Pc = Probability of Crossover = 0.95

Population Size = 50

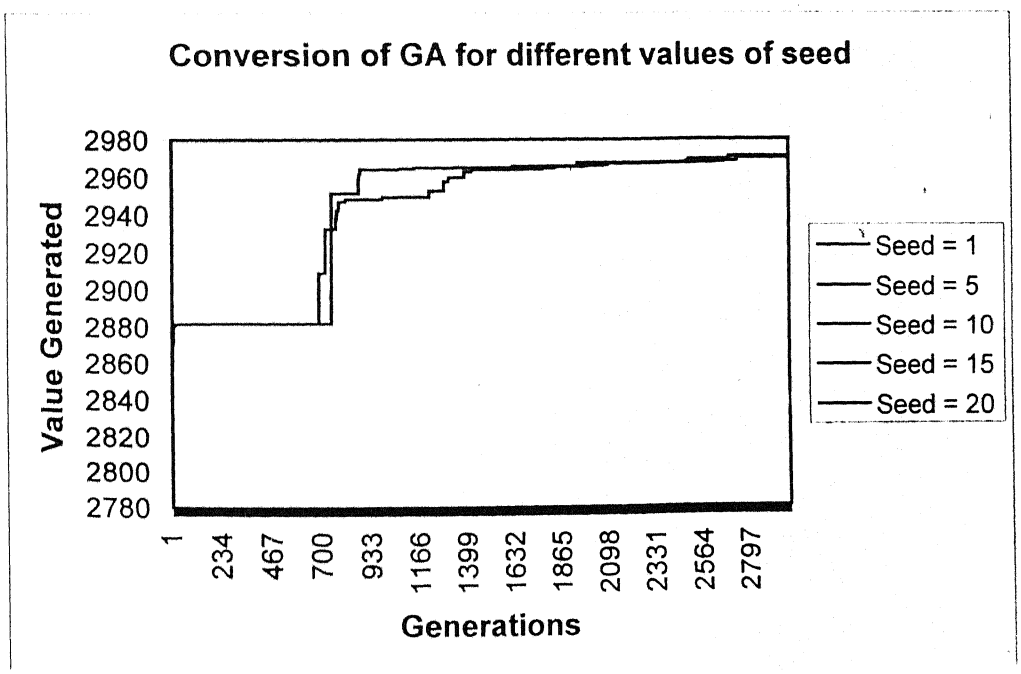
Generations = 5000

7.4 Graphs

7.4.1 Conversion Graph for GA :



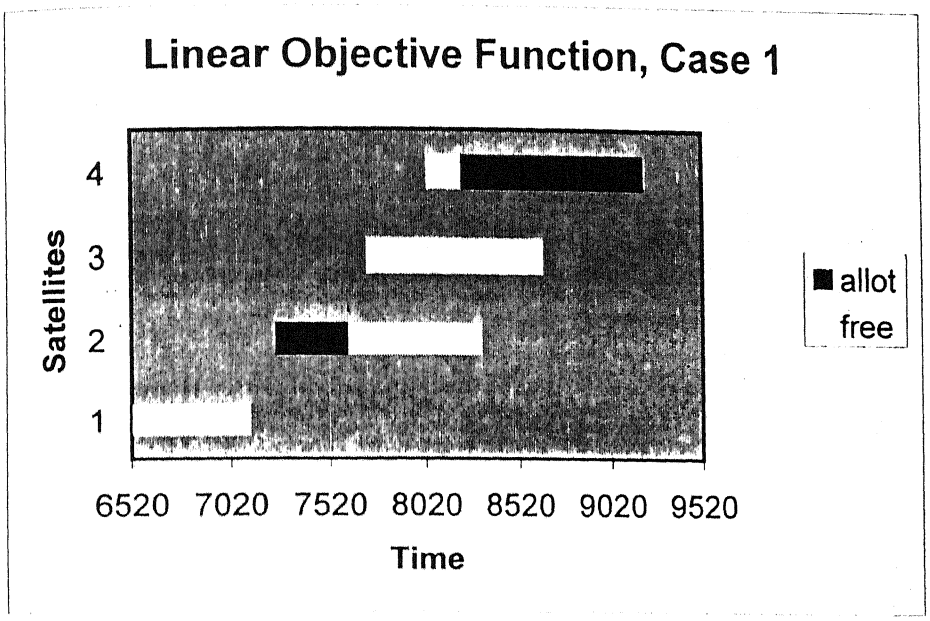
7.4.2 Conversion Graph for different values of seed:



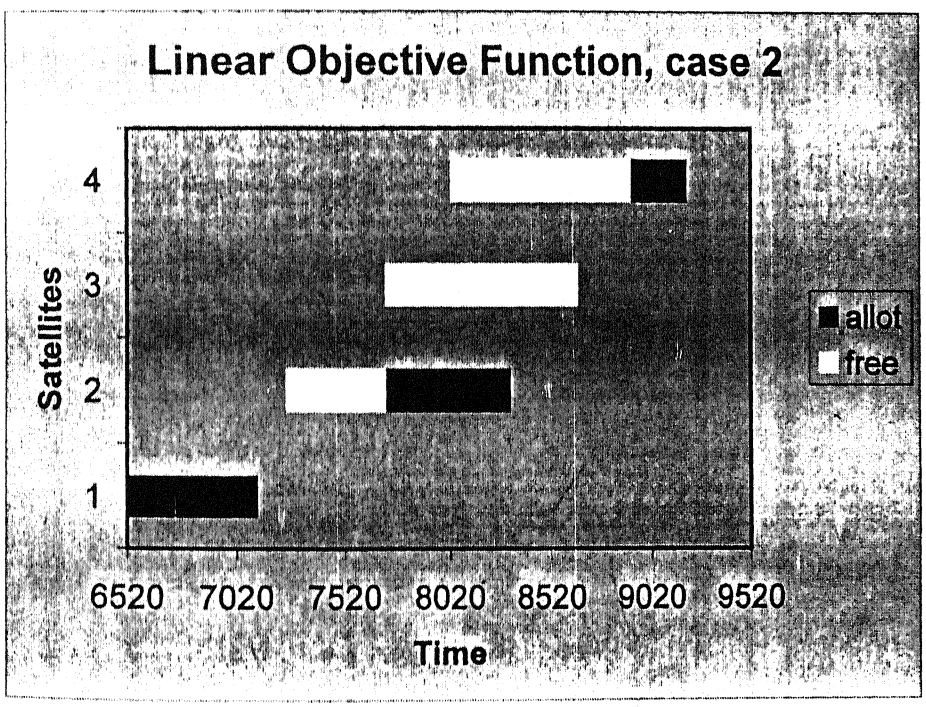
7.4.3 Satellite Visibility support Graphs for GA:

Linear Objective function:

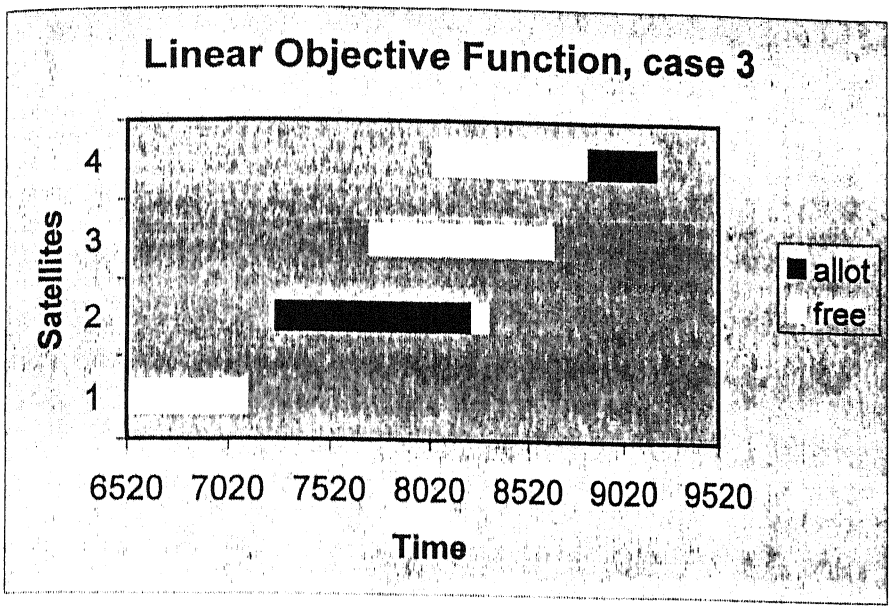
Problem case 1: v1=1 v2=2 v3=3 v4=4



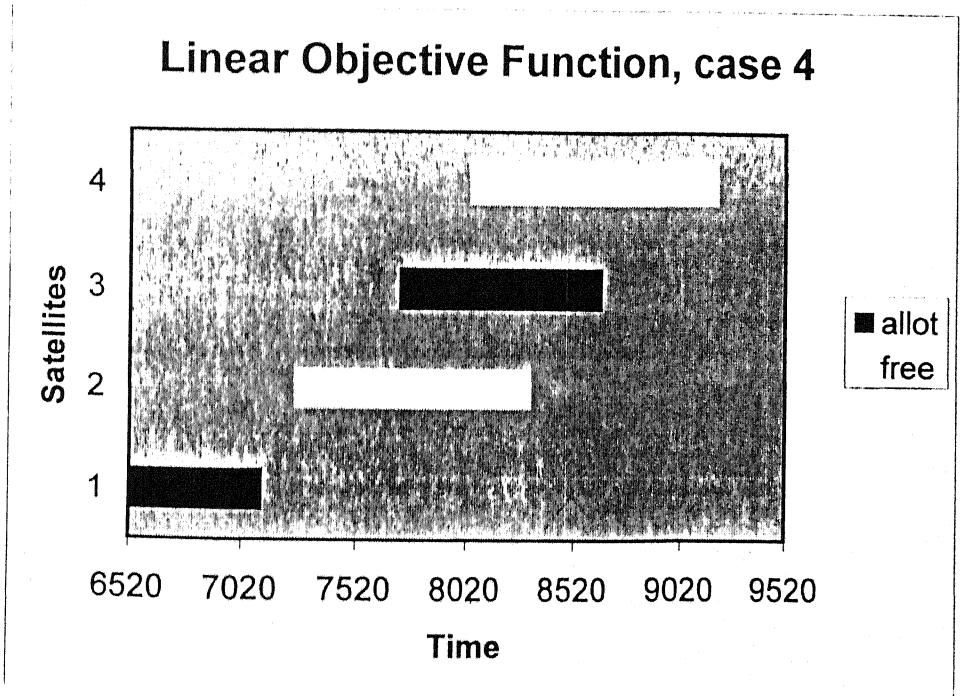
Problem case 2: v1 = 4 v2= 3 v3= 2 v4=1



Problem case 3: v1=1 v2=4 v3=3 v4=2

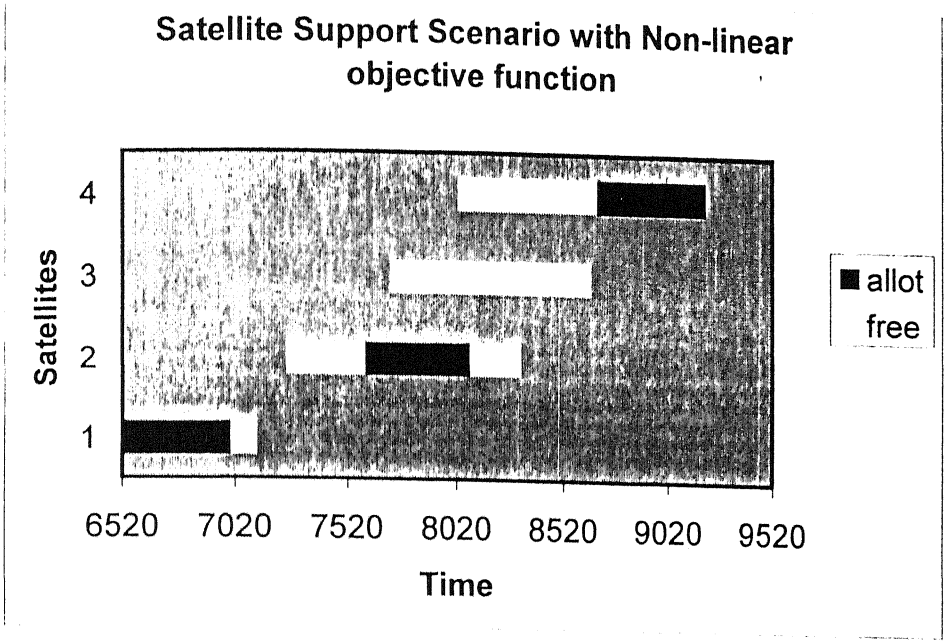


Problem case 4: v1=1 v2=3 v3=4 v4=2

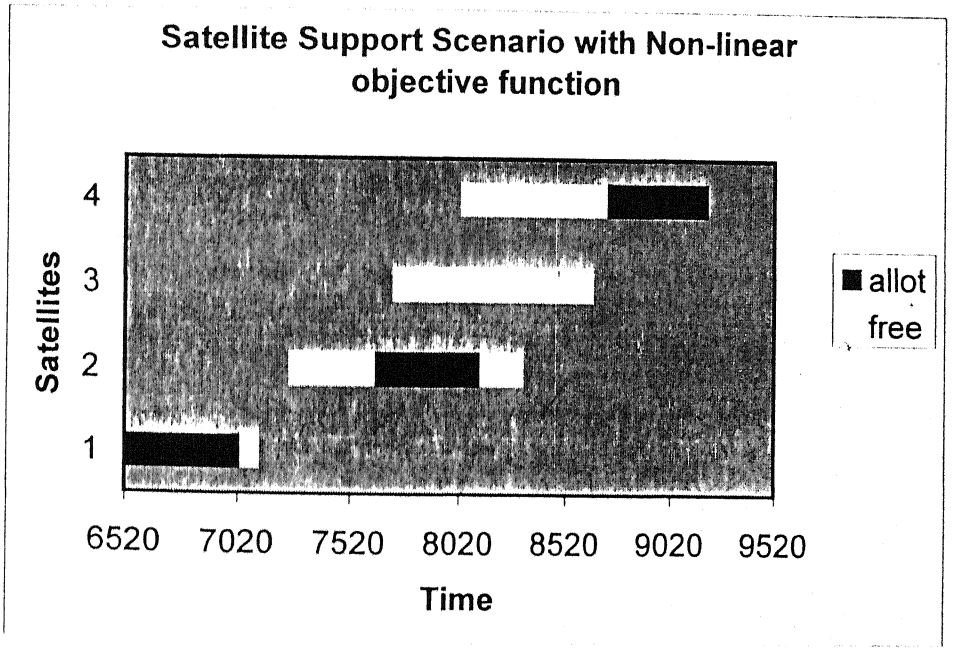


Non Linear Objective Function:

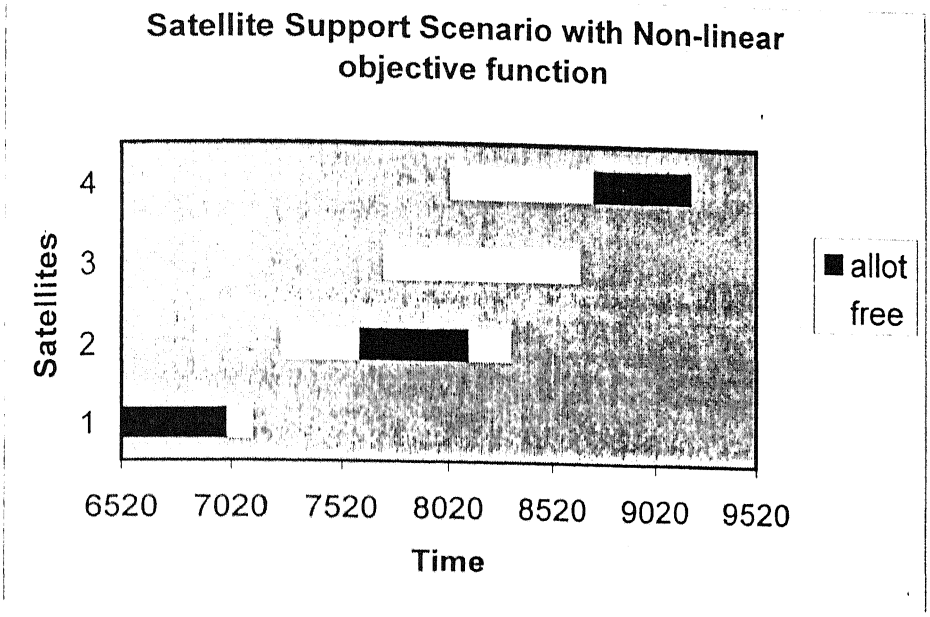
Problem case 1: v1=1 v2=2 v3=3 v4=4



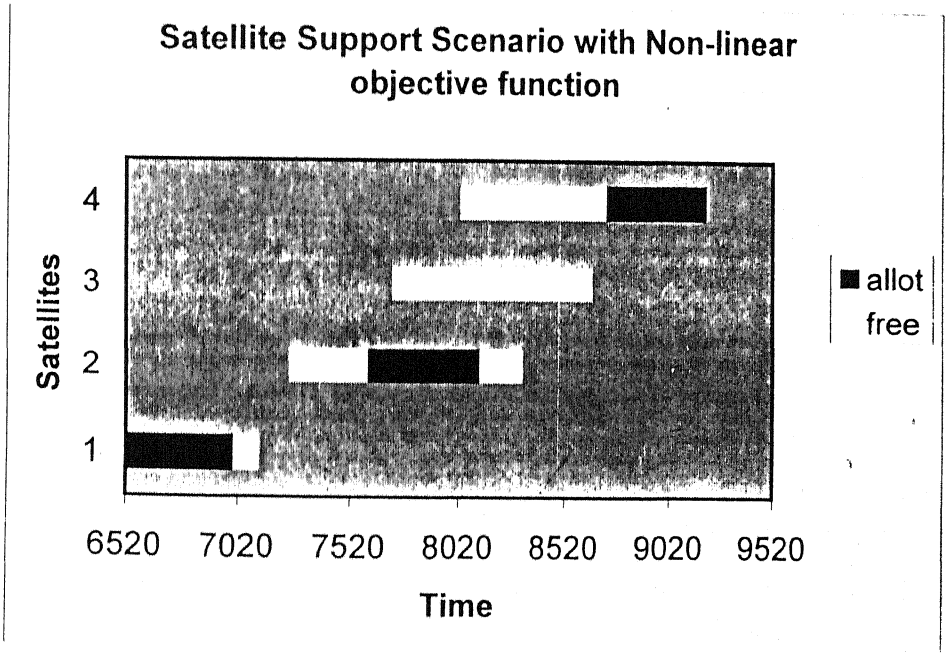
Problem case 2: v1=4 v2=3 v3=2 v4=1



Problem case 3: v1=1 v2=4 v3=3 v4=2



Problem case 4: v1=1 v2=3 v3=3 v4=2



Chapter 8
Comparison

In this chapter we compare the results obtained by applying previously discussed algorithms such as Greedy Search, Tabu Search, Simulated Annealing and the GA by Kumar and Bagchi [3]. We take here a sample problem with four scenarios, (each obtained by changing the weight factor of satellites that are involved in visibility clash.)

8.1 The Results of comparison:

The problem we consider here is a real life clash situation in which four satellite visibilities clash. The visibility start times and visibility end times are as follows.

vis_start 1 = 6520 vis_end 1 = 7115

vis_start 2 7244 vis_end 2 8318

vis_start 3 = 7712 vis_end 3 = 8650

vis_start 4 = 8027 vis_end 4 = 9196

The cases described below are obtained by changing the priority factor of different satellites. These factors determine the total value generated from the supports provided. The different priority factor combinations considered are as follows.

Case 1: v1 =1	v2=2	v3=3	v4=4
Case2: v1 =4	v2=3	v3=2	v4=1
Case3: v1 =1	v2=4	v3=3	v4=2
Case 4: v1 =1	v2=3	v3=4	v4=2

We considered two different value generating objective functions. The first is linear (applied for Payload operations) and the other is non-linear (applied for TTC type satellite operations).

Linear Objective function:

Maximise

$F = \sum_{\text{over all } i}\{(e_i - s_i)*v_i\}$ **.. (for PL operations)**

Non-Linear Objective function :

Maximise

$$F = \sum_{\text{over all } i}\{A_i + (1-e^{-(ci-si)*.00958} * v_i)/.00958\} \quad \text{..(for TTC operations)}$$

where

A_i= initial support time * v_i

e_i = end of support of satellite i

s_i = start of support of satellite i

v_i = priority weight of satellite i

Table 8.1

Best Solutions obtained for the four different algorithms:

Problem Case no.	Linear objective function				Non-Linear objective function			
	GA	Greedy Search	Tabu Search	Simulated Annealing	GA	Greedy Search	Tabu Search	Simulated Annealing
1	4670	4670	4670	4670	3478.67	3415.99	3481.79	3481.79
2	4467	4467	4467	4467	3958.48	3855.55	3962.04	3962.04
3	4670	4670	4670	4670	3481.01	3415.99	3481.79	3481.79
4	4344	4344	4344	4344	2969.49	2881.89	2971.344	2971.344

Table 8.2

Computer time (in milliseconds) taken by four different algorithms:

The algorithms were run on a Pentium III , 800 MHz processor machine with 128 MB RAM and 20 GB HDD.

Problem case no	Linear objective function				Non-Linear objective function			
	GA	Greedy Search	Tabu Search	Simulated Annealing	GA	Greedy Search	Tabu Search	Simulated Annealing
1	540	390	390	4730	7460	490	490	4780
2	450	330	390	5540	7523	550	600	5760
3	480	330	390	4510	7765	440	500	4780
4	560	330	330	4510	7612	440	600	4720

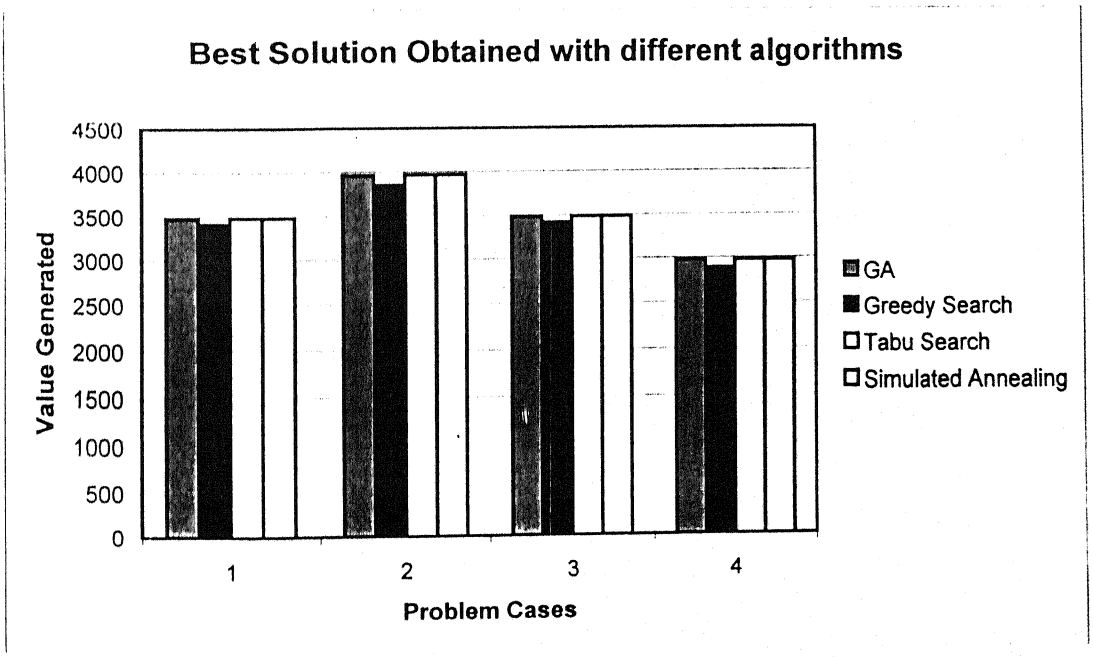
Table 8.3 Support Scenario generated by four different algorithms:

Problem Case: v1 =1, v2= 2, v3= 3, v4= 4

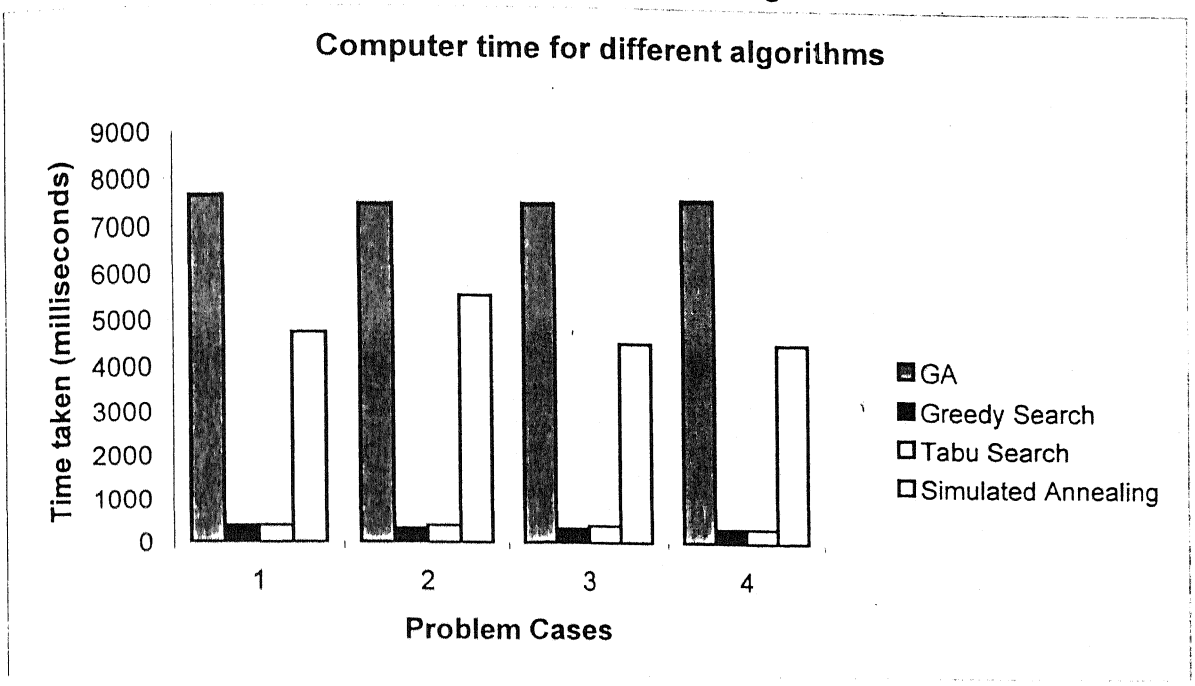
Algorithm	Support Start Time s_i	Support End Time e_i	Total Support given: $e_i - s_i$
GA	6520	7000.01	480.01
	7600.01	8082.5	482.489
	7712	7712	0
	8482.5	9196	513.5
Greedy Search	6520	6520	0
	7244	7884	640
	7712	7712	0
	8484	9196	712
Tabu Search	6520	7000	480
	7600	8080	480
	7712	7712	0
	8680	9196	516
Simulated Annealing	6520	7000	480
	7600	8080	480
	7712	7712	0
	8680	9196	516

8.2 Graphs

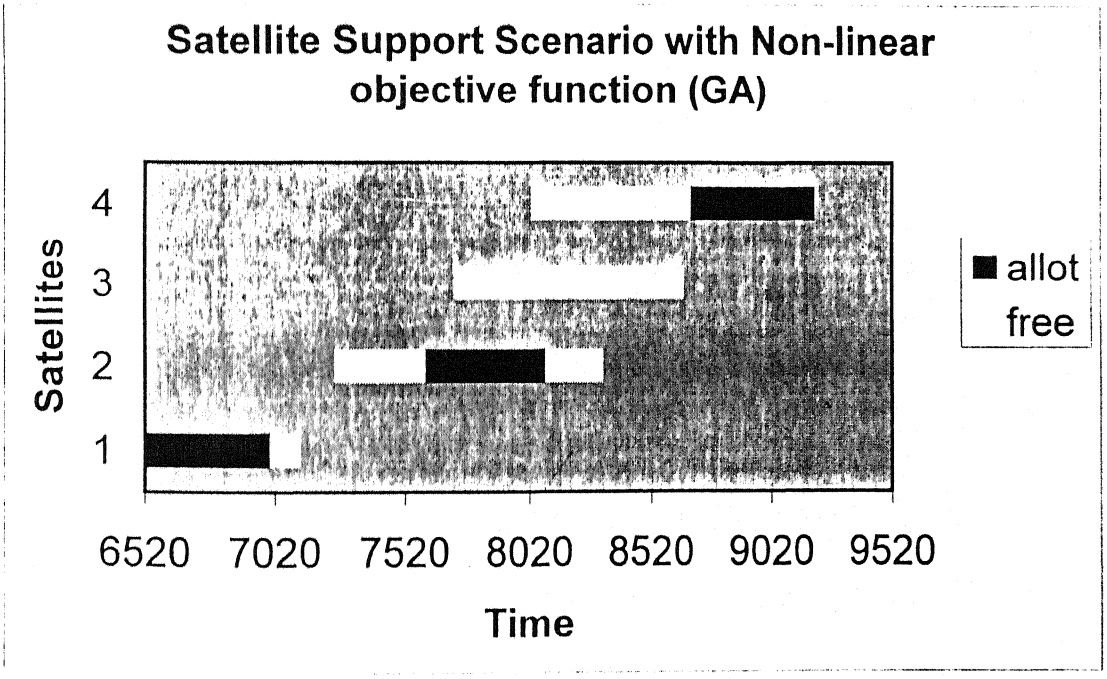
8.2.1 Graph for best solution obtained with different algorithms:

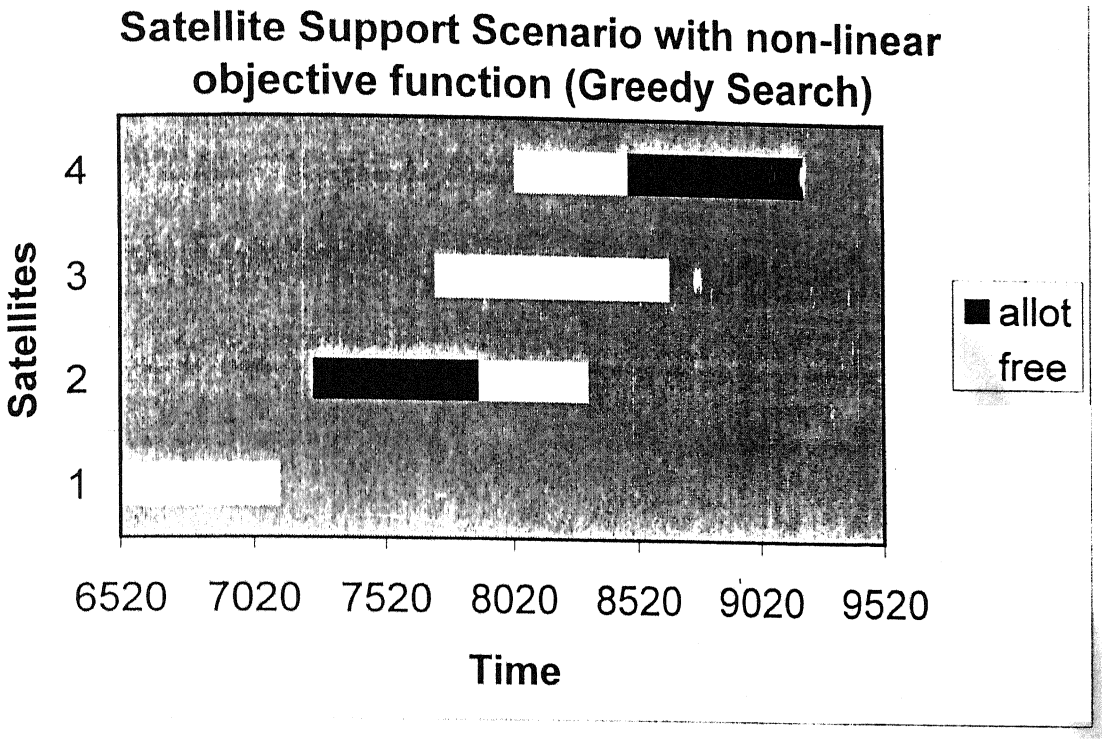


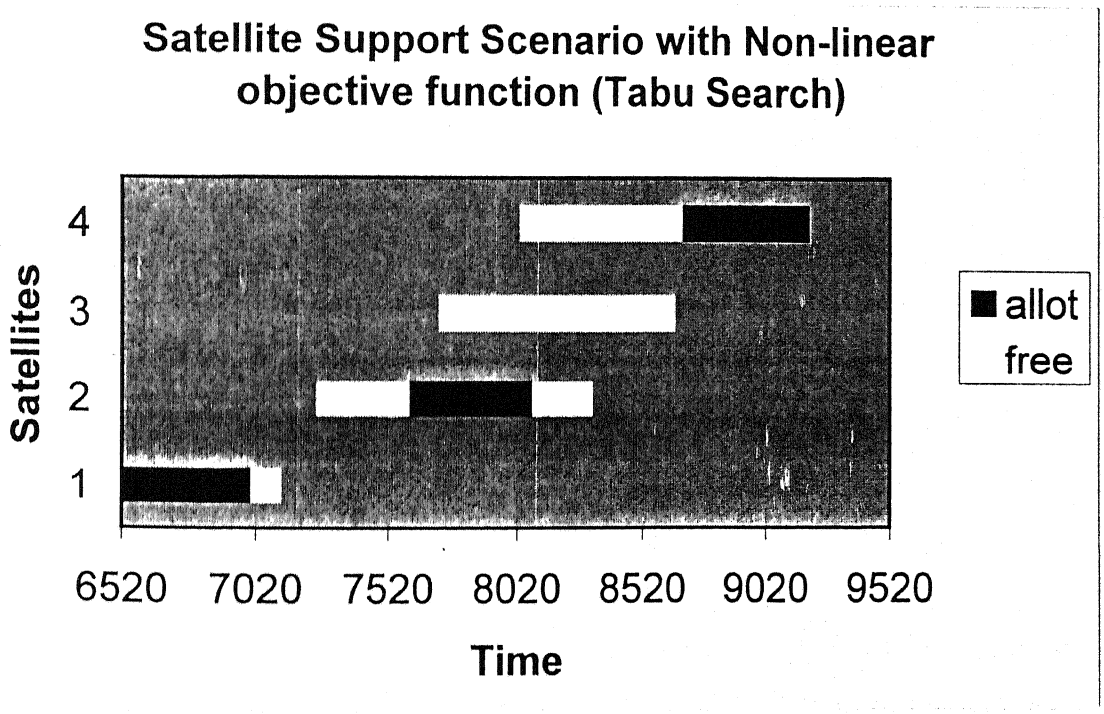
8.2.2 Graph for Computer time taken by different algorithms:

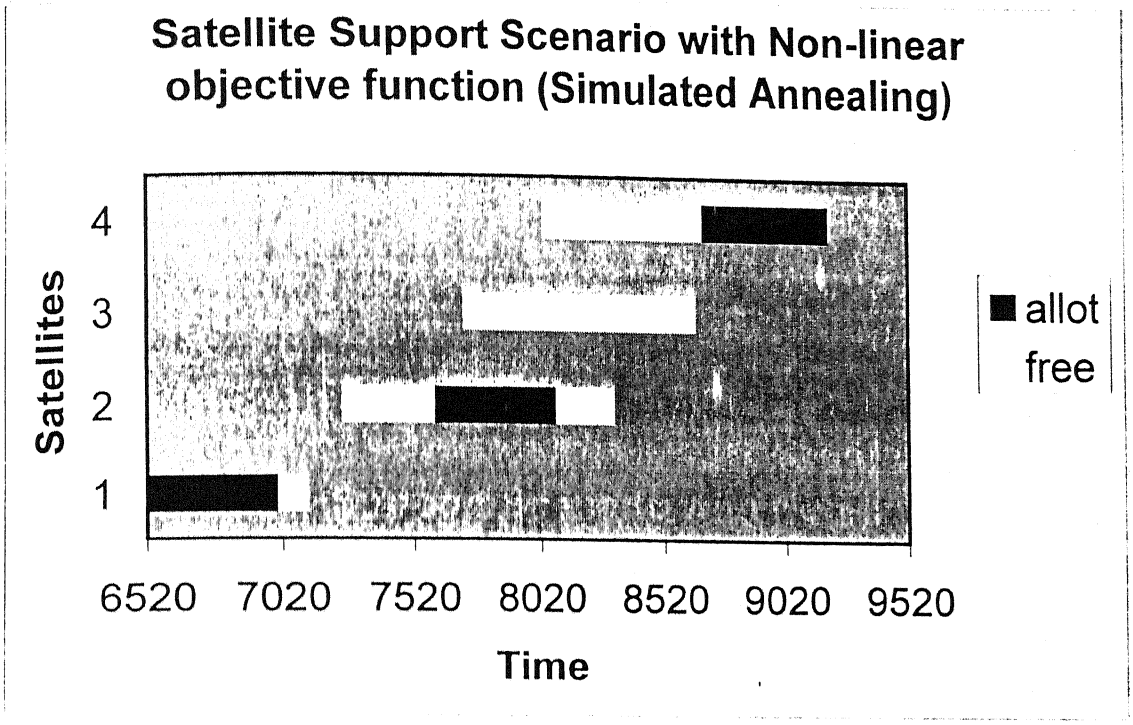


8.2.2 Graphs for Satellite Support Scenario obtained with different algorithms:




Satellite Support Scenario with non-linear objective function (Greedy Search)
Satellites
4
3
2
1
6520
7020
7520
8020
8520
9020
9520
Time
allot
free


Satellite Support Scenario with Non-linear objective function (Tabu Search)
Satellites
4
3
2
1
6520
7020
7520
8020
8520
9020
9520
Time
allot
free



8.3 Analysis:

8.3.1 Best Results Obtained:

The results obtained show that there is no difference in the best solution obtained by four algorithms when we use the linear objective function for evaluation (i.e. Payload visibility support). We find some minor difference in the best value obtained when we use the non-linear objective function. (i.e. for TTC visibility support). First we consider the Greedy Algorithm. As mentioned earlier, if number of initial solutions is increased, it increases the likelihood of reaching to global optima. Here though we are using a large number of initial solutions (Chapter 4, Eq. 4.2.5), the shape of the search space is very complex and hence the Greedy Algorithm terminates in a local optima (which is very near to the global optima). But most of the times we find that this algorithm also works fine to get the global optima for this problem.

The other three algorithms, GA, Tabu Search and Simulated Annealing are well known to find a global optimum solution. The critical thing is setting of parameters for these algorithms. We, as described before, optimised the parameters using Design of Experiments. As shown above all these three algorithms are able to find global optimum solution.

8.3.2 Computer Time:

We tested the algorithms on a IBM machine having Pentium III processor with 800Mhz speed, 128 RAM, 20 HDD. The computer time taken by Tabu Search and Greedy Search is almost a second or less than that for resolving a visibility clash of four satellites. However, the time taken by Simulated Annealing and GA is comparatively more. This can be explained as follows.

The Simulated Annealing algorithm needs a number of iterations at each temperature value and the algorithm proceeds with a small decrement in temperature at each step. (slow cooling). Thus, to obtain final result one has to make comparatively a large number of iterations in SA. Hence, the computer time required is more as compared to Tabu Search or Greedy Search algorithms.

In case of GA, we have to run the algorithm for as much as 2000 generations to get to the best solution obtained by Tabu Search and Simulated Annealing. This is because the shape of the search space is very complex. Hence GA also takes comparatively more time to reach the global optimum solution.

The interesting thing to note here is that Tabu Search is able to reach the global optima in a comparatively less amount of time because of the diversification criteria used. We select different combinations of initial solutions (as explained earlier) and run the algorithm with these solutions as starting solutions, hence the global optimum solution is obtained comparatively quickly.

Chapter 9
Conclusions

The problem of scheduling satellite operations is a complex process due to a sizable number and type of *constraints* imposed from operations, satellites, stations and controller, and non-linear objectives and constraints. Also, it is experienced that this problem of satellite scheduling is difficult to be mapped to existing solution paradigms in the literature, such as linear or integer programming. Thus, conventional methods cannot be used to produce a satellite operations support schedule.

One of the critical things in TTC operations scheduling is the designing of objective function. Because of many qualitative decisions involved, it is very difficult to map all the factors in the objective function. We have developed a exponentially decaying function which takes into consideration most of the factors discussed with ISRO and which also tackles the issue of dynamic priority.

In this study we tested four different algorithms on the same problem. Though all the three metaheuristic algorithms implemented here, viz, GA, Tabu Search and Simulated Annealing and also Greedy Search reached to the optimal or near to optimal solution, we could see that Tabu Search reached the global optimum in comparatively less amount of time. If we look at the results of the three algorithms more closely we find the following things,

In case of Greedy Search, the algorithm performed very well when we used the linear objective function, but many times the algorithm was not able to reach the global optimum solution when we used non-linear objective function. This shows that this algorithm is mostly applicable for the clash resolution of the payload type operations support where the objective function is linear.

Genetic Algorithm performed very well on both type of objective functions, i.e linear and non-linear. In case of non-linear objective function, GA took more computer time to reach to the global optimum solution than other two metaheuristic algorithms (Tabu Search and Simulated Annealing) tested.

Tabu Search turned out to be the best algorithm among the other two as it reached global optimum solution in case of both the linear and the non-linear objective function. Furthermore, it took comparatively much less time to reach the global optimum solution than other two algorithms.

In case of Simulated Annealing, we find that the algorithm reached the global optima in both the cases of linear and non-linear objective function. The algorithm took computer time that is more than Tabu Search but less than GA.

This work establishes that Tabu Search works more effectively than Genetic Algorithm and Simulated Annealing in resolving clashed visibilities for TTC type of operations, the *key* difficulty in satellite scheduling.

Bibliography

[1] Agnese J C and Brousse P(1998) "Scheduling Techniques for a constellation of Visibilities," *Spaceflight Dynamics*; Proc AAS/ GSFC International Symposium, Greenbelt, MD, May.

[2] Bagchi T P (1999), *Multiobjective scheduling by Genetic Algorithms*, Kluwer .

[3] Bagchi T P, Kumar S (2000), "LEO Satellite Visibility Clash Resolution by Genetic Algorithms", OR-2000 XXXIII Annual Convention of the OR Society of India, Ahmedabad. Operations Management Conference 4, Dec 22-24, 2000. Chennai.

[4] Bottcher A, Jahn A, Lutz E and Werner M (1993),"Analysis of basic system parameters of communication networks based on low earth orbit satellites."

[5] Deb K (1995) *Optimisation for Engineering Design: Algorithms and Examples* New Delhi, Prentice-Hall.

[6] Galiber F and Dimitrov S A (1998)" A Satellite Mission Planning Framework."

[7] Glover F, Laguna M (1997) *Tabu Search*, Kluwer Publication

[8] Goldberg D E (1989), *Genetic Algorithms in search, Optimization and Machine Learning*. Addition-Wisley, Reading Mass, USA.

[9] GREAS, Generic Resource, Event and Activity Scheduler. http://www.veridian.com/Products/grweb2.asp

[10] Johnston M D and Miller G D, "SPIKE: Intelligent Scheduling of Hubble Space Telescope Observations" http://www.stsci.edu/apsb/doc/papers-and-meetings/93-Intelligent-Scheduling/spike/spike-chapter3.html

[11] Kennedy D J, Parnell G S and Rowell W F (1988) "An Expert System for Scheduling Satellite Supports", *Interfaces* 18:6 Nov-Dec 1988, pp. 28-34

[12] Kirkpatrick S, Gelatt C D, Vecchi M P Jr (1983), "Optimization by Simulated Annealing", *Science*, 13 May 1983, Volume 220, Number 4598.

[13] Laarhoven P J M and E.H.L. Arts (1987), *Simulated Annealing: Theory and Applications.*

[14] Learning Optimisation from Nature, Simulated Annealing, ChemoWeb, Chemometrics Research Group, Naval Research Laboratory, Washington DC.
http://chemdiv-www.nrl.navy.mil/6110/sensors/chemometrics/optsa.html

[15] Mostert S and Milne G W (2000), "Sunsat: Solutions for Remote Sensing".
http://sunsat.ee.sun.ac.za/sspapers/sats.95/sats.html

[16] Mukherjee A(1998), "Skirmishes in the sky", *Computers Today*, March 1998.

[17] Rao J D, Soma P, Padmashree G S (1998), "Multisatellite scheduling system for Leo Satellite Operations."
http://www.nasda.go.jp/SpaceOps/paper98/track2/2b002.pdf

[18] Thiel S U, "Tabu Search, Overview of Modern Heuristic Methods."
http://egeria.cs.cf.ac.uk/User/S.U.Thiel/ra/subsection3_7_2.html

[19] Wolfe W J and Soresen S E (2000), "Three scheduling Algorithms Applied to the Earth Observing Systems Domain," *Management Science*, Vol. 46, No 1, January 2000 pp. 148-168.

APPENDIX 1
COMMON PROGRAMS

1.Evaluation.java

```
import java.io.*;
import java.util.*;
import java.lang.*;
class Variable
{
	int[] S,E,s,e,s_w,s_n;

	Variable()
	{}

	Variable(int[] vis_start,int[] vis_end,int[] allocate_start,int[]
	allocate_end,int[] sat_wt,int[] sat_no)
	{
		S = new int[vis_start.length];
		s = new int[allocate_start.length];
		E = new int[vis_end.length];
		e = new int[allocate_end.length];
		s_w = new int[sat_wt.length];
		s_n = new int[sat_no.length];

		for(int i=0;i<S.length;i++)
		{
			S[i]=vis_start[i];
			s[i]=allocate_start[i];
			E[i]=vis_end[i];
			e[i]=allocate_end[i];
			s_w[i] = sat_wt[i];
			s_n[i] = sat_no[i];
		}
	}
}


class Evaluation
{
	Evaluation()
	{}
	public static double evaluation(Variable vb)
	throws Exception
	{
		Variable vrb = new Variable();
		vrb=vb;
		double result;
		int []w;
		double []A;
		int min_support = 480;

		w = new int[vrb.s_w.length];
		A = new double[vrb.s_w.length];
```

```
            for(int i=0;i<vrb.s_w.length;i++)
            {
                  w[i] (vrb.s_w[i]);
                  A[i]= (min_support*w[i]);
            }
            result=0;
            int flag;
            for(int i=0;i<vrb.s_w.length;i++)
            {
                  if((vrb.e[i]-vrb.s[i])>=min_support)
                  flag = 1;
                  else
                  flag = 0;
                  result=result+(A[i]+(((1-(Math.pow(2.718281,(-
                  ((vrb.e[i]-vrb.s[i]-
                  min_support)*(.00958)))))))*w[i])/.00958))*flag;
            }
      return result;
      }
}
```

---

2. Feasibility.java

```
class Feasibility
{
      Feasibility()
      {}
      public static boolean feasibility(Variable vr)
      {
            double min = 480,max = 983,recon =600;
            Variable vrb = new Variable();
            vrb =vr;
            boolean flag = false;

            for(int i=0;i<vrb.S.length;i++)
            {
                  flag=false;

            if((vrb.s[i]<=vrb.e[i])&&(vrb.e[i]<=vrb.E[i])
                  &&(vrb.S[i]<=vrb.s[i]))
                  {
                  if((((vrb.e[i]-vrb.s[i])>=min)||((vrb.e[i]-
                  vrb.s[i])==0))&&((vrb.e[i]-vrb.s[i])<=max))
                  {
                  if((vrb.e[i]-vrb.s[i])>=min)
                  {
                  if(i!=(vrb.S.length-1))
                  {
                  if((vrb.s[i+1]-vrb.e[i])>=recon)
                        flag=true;
                  else
                        break;
                  }
                  else
                  flag=true;
```

```
                }
                else
                {
                if(i!=(vrb.S.length-1))
                {
                      if((vrb.s[i+1]-vrb.e[i])>=0)
                      flag=true;
                      else
                      break;
                }
                else
                flag=true;
                }
                }
                else
                break;
                }
                else
                break;
          }
          return flag;
     }
}
```

---

3. FineTuning.java

```
import java.lang.*;
import java.io.*;
import java.util.*;

class FineTuning
{
      FineTuning()
      {}

      public static Variable fineTuning(Variable_array v_array)
      throws Exception
      {
            Vector v;
            Vector v1;
            Vector v2;

            Variable va = new Variable();
            Variable_array varray = new Variable_array();
            double optimum=0;
            varray = v_array;
            for(int m=0;m<varray.vrba.length;m++)
            {
                  double
                  finalmax=0,previous=0,next=0,index_previous=0,index_n
                  ext=0;
                  boolean flag=true,flag1,flag2=true;
                  double answer=0,answer1;
                  double []temp;
                  int iter=10000,prob,count1=0;
```

```
Random rm = new Random((long)19.00);
Variable vrb= new Variable();
Variable vrb_final = new Variable();
Feasibility fb = new Feasibility();
Evaluation ev = new Evaluation();
Neighborhood1 nh = new Neighborhood1();
Variable_array  vr_array = new Variable_array();

vrb=varray.vrba[m];
for(int j=0;j<iter;j++)
{
      vr_array = nh.neighborhood1(vrb);
      v = new Vector();
      v1 = new Vector();

      flag = fb.feasibility(vrb);
      if(flag)
      {
      answer=ev.evaluation(vrb);
      if(answer>finalmax)
      {
            finalmax=answer;
            vrb_final=vrb;
      }
      previous=next;
      next = finalmax;
      if(previous==next)
      {
            if(flag2)
            {
            flag2=false;
            index_previous = j;
            index_next=j;
            }
            count1++;
            index_next=j;
      }
      if((count1==20)&&((index_next-
      index_previous)==19))
            break;
      else
      {
            if(count1==20)
            {
            flag2=true;
            count1=0;
            }
      }

      for(int i=0;i<vr_array.vrba.length;i++)
      {
            flag1 = fb.feasibility(vr_array.vrba[i]);
            if(flag1)
            {

answer1=ev.evaluation(vr_array.vrba[i]);
```

```
                        v.addElement(new Double(answer1-
answer));
                        v1.addElement(new Integer(i));
                    }
                }
                Enumeration vEnum = v.elements();
                double max=-12000;
                int element=0,index=0,index1=0;
                v2 = new Vector();

                temp = new double[v.size()];
                while(vEnum.hasMoreElements())
                {
                temp[element] =
                Double.parseDouble(vEnum.nextElement().toString
                ());
                if(temp[element] > max)
                {
                      if(temp[element]>max)
                      {
                      max=temp[element];
                      index = element;
                      }
                else
                {
                      max=temp[element];
                      v2.addElement(new Integer(element));
                }
                }
                element++;
                }
                if(v1.isEmpty()==false)
                {
                if(v2.isEmpty()==true)
                {
                vrb =
                vr_array.vrba[Integer.parseInt(v1.elementAt(ind
                ex).toString())];
                }
                else
                {
                float interm =((rm.nextFloat())*(v2.size()+1));
                prob=(int)interm;
                if(prob==v2.size())
                index1=index;
                else
                index1=Integer.parseInt(v2.elementAt(prob).toSt
                ring());
                vrb =
                vr_array.vrba[Integer.parseInt(v1.elementAt(ind
                ex1).toString())];
                }
          }
          else
          break;
          }
          else
```

```
                break;
            }
            if(flag)
            {
            if(finalmax>=optimum)
            {
                optimum=finalmax;
                va=vrb_final;
            }
            }
            else{}
            }
            return(va);
        }
}
```

4.InitialSolution.java

```
import java.io.*;
import java.util.*;
import java.lang.*;

class InitialSolution
{
      static int [] vis_start0;
      static int[] vis_end0;
      static int [] sat_wt0;
      static int [] sat_no0;
      static int min=480;
      static int recon 600;
      static int max 983;
      static int[][] binary;

      public static Variable_array initialsolution(InputClass iclass)
      throws Exception
      {
            Variable [] vra;
            Variable_array varray;

            vis_start0 = new int[(iclass.S.length)];
            vis_end0 = new int[(iclass.E.length)];
            sat_wt0 =  new int[(iclass.s_w.length)];
            sat_no0 =  new int[(iclass.s_n.length)];

            for(int i=0;i<iclass.S.length;i++)
            {
                  vis_start0[i] = iclass.S[i];
                  vis_end0[i] = iclass.E[i];
                  sat_wt0[i] = iclass.s_w[i];
                  sat_no0[i] = iclass.s_n[i];
            }

            int n=vis_start0.length;
            int b =0,c=0;
            double p = Math.pow(2.0,n);
```

```
        vra = new Variable[(int)p];
        int [][]binary = new int[(int)p][n];

        for(int i=0;i<(int)p;i++)
        {
                b=c;
                for(int j=0;j<n;j++)
                {
                        binary[i][j]=b%2;
                        b=b/2;
                }
        c++;
        }

for(int i=0;i<p;i++)
{
        int count=0;
        for(int j=0;j<n;j++)
        {
                if(binary[i][j]==1)
                {
                        count++;
                }
        }

        int []vis_start = new int[count];
        int []vis_end = new int[count];
        int []all_start = new int[count];
        int []all_end = new int[count];
        int []sat_wt = new int[count];
        int []sat_no = new int[count];

        int s=0;
        for(int j=0;j<n;j++)
        {
                if(binary[i][j]==1)
                {
                        vis_start[s] = vis_start0[j];
                        vis_end[s] = vis_end0[j];
                        sat_wt[s] = sat_wt0[j];
                        sat_no[s] = sat_no0[j];
                        s++;
                }
        }

        int k=count-1;
        for(int j=n-1;j>=0;j--)
        {

                if(k==0)
                {
                if(binary[i][j]==1)
                 {
                        all_start[k]=vis_start[k];
                        all_end[k]=all_start[k]+min;
                        k--;
                }
```

```
                }
                else
                {
                        if(k==(count-1))
                        {
                        if(binary[i][j]==1)
                        {
                                all_end[k]=vis_end[k];
                                all_start[k]=all_end[k]-min;
                                k--;
                        }
                }
                else
                {
                if(binary[i][j]==1)
                {
                        all_end[k]=Math.min((all_start[k+1]-
                        recon),vis_end[k]);
                        all_start[k]=all_end[k]-min;
                        k--;
                }
                }
        }
    }

    vra[i] = new
Variable(vis_start,vis_end,all_start,all_end,sat_wt,sat_no);
}

varray =new Variable_array(vra);
return(varray);
}
}
```

5.InitialSolution1.java

```
import java.io.*;
import java.util.*;
import java.lang.*;

//this class is useful for greedy heuristic..
//this generates more than 2^ n initial solutions.
//the formula is given below
//formula = 2^n+(sigma i=3 to n)(nCi*2^(i-2)-nCi)

class InitialSolution1
{
    InitialSolution1()
    {}
    static int [] vis_start0;
    static int [] vis_end0;
    static int [] sat_wt0;
    static int [] sat_no0;
    static int min=480;
    static int recon=600;
```

```
static int max=983;

 public static Variable_array initialsolution1(InputClass iclass)
 throws Exception
 {
      Variable [] vra;
      Variable_array varray;
      double p,d=0;
      int n,b=0,c=0;
      vis_start0 = new int[(iclass.S.length)];
      vis_end0 = new int[(iclass.E.length)];
      sat_wt0 =  new int[(iclass.s_w.length)];
      sat_no0 =  new int[(iclass.s_n.length)];

      for(int i=0;i<iclass.S.length;i++)
      {
            vis_start0[i] = iclass.S[i];
            vis_end0[i] = iclass.E[i];
            sat_wt0[i] = iclass.s_w[i];
            sat_no0[i] = iclass.s_n[i];
      }

      n=vis_start0.length;
      p = Math.pow(2.0,n);

      if(n<3)
      d=p;
      else//formula = 2^n+(sigma i=3 to n)(nCi*2^(i-2)-nCi)
      {
            if(n==3)
            d=9;
            if(n==4)
            d=23;
            if(n==5)
            d=64;
            if(n==6)
            d=186;
            if(n==7)
            d=551;
            if(n==8)
            d=1645;
     }

     int [][] binary = new int[(int)p][n];
     int [][] binary1 = new int[(int)d][n];

     for(int i=0;i<(int)p;i++)
     {
           b=c;
           for(int j=0;j<n;j++)
           {
                 binary[i][j]=b%2;
                 b=b/2;
           }
    c++;
     }
```

```
int x=0;
double count1;
for(int i=0;i<(int)p;i++)
{
	count1=0;
	for(int j=0;j<n;j++)
	{
		if(binary[i][j]==1)
		count1++;
	}
	if(count1<3)
	{
		for(int j=0;j<n;j++)
		binary1[x][j]=binary[i][j];
		x++;
	}
	else
	{
		for(int k=0;k<Math.pow(2,(count1-2));k++)
		{
			for(int j=0;j<n;j++)
			binary1[x][j]=binary[i][j];
			x++;
		}
	}
}
boolean flag=true;
int i_4=0;
int i_5=0;
int i_6=0;
int i_7=0;
int i_8=0;

vra = new Variable[(int)d];

int new_index=0;
for(int i=0;i<d;i++)
{
	int count=0;
	for(int j=0;j<n;j++)
	{
		if(binary1[i][j]==1)
		count++;
	}

	int []vis_start = new int[count];
	int []vis_end = new int[count];
	int []all_start = new int[count];
	int []all_end = new int[count];
	int []sat_wt = new int[count];
	int []sat_no = new int[count];

	int s=0;
	for(int j=0;j<n;j++)
	{
		if(binary1[i][j]==1)
		{
```

```
                vis_start[s] = vis_start0[j];
                vis_end[s] = vis_end0[j];
                sat_wt[s] = sat_wt0[j];
                sat_no[s] = sat_no0[j];
                s++;
        }
}

switch(count)
{
case 0: break;
case 1:
        all_start[0]=vis_start[0];
        all_end[0]= all_start[0]+min;
        break;
case 2:
        all_start[0]=vis_start[0];
        all_end[0]= all_start[0]+min;
        all_end[1]=vis_end[1];
        all_start[1]=all_end[1]-min;
        break;
case 3:
        if(flag)
        {
        all_start[0]=vis_start[0];
        all_end[0]= all_start[0]+min;
        all_end[2]=vis_end[2];
        all_start[2]=all_end[2]-min;
        all_start[1]=Math.max((all_end[0]+recon),
        vis_start[1]);
        all_end[1]= all_start[1]+min;
        flag=false;
        break;
        }
        else
        {
        all_start[0]=vis_start[0];
        all_end[0]= all_start[0]+min;
        all_end[2]=vis_end[2];
        all_start[2]=all_end[2]-min;
        all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
        all_start[1]= all_end[1]-min;
        flag=true;
        break;
        }
case 4:
        if(i_4==0)
        {
        all_start[0]=vis_start[0];
        all_end[0]= all_start[0]+min;
        all_end[3]=vis_end[3];
        all_start[3]=all_end[3]-min;
        all_start[1] = Math.max((all_end[0]+recon),
        vis_start[1]);
        all_end[1]= all_start[1]+min;
        all_start[2] = Math.max((all_end[1]+recon),
        vis_start[2]);
```

```
        all_end[2]= all_start[2]+min;
        i_4++;
        break;
        }
        if(i_4==1)
        {
        all_start[0]=vis_start[0];
        all_end[0]= all_start[0]+min;
        all_end[3]=vis_end[3];
        all_start[3]=all_end[3]-min;
        all_start[1]=Math.max((all_end[0]+recon),
        vis_start[1]);
        all_end[1]= all_start[1]+min;
        all_end[2]=Math.min((all_start[3]-recon),vis_end[2]);
        all_start[2]= all_end[2]-min;
        i_4++;
        break;
        }
        if(i_4==2)
        {
        all_start[0]=vis_start[0];
        all_end[0]= all_start[0]+min;
        all_end[3]=vis_end[3];
        all_start[3]=all_end[3]-min;
        all_end[1]=vis_end[1];
        all_start[1]= all_end[1]-min;
        all_start[2]=Math.max((all_end[1]+recon),
        vis_start[2]);
        all_end[2]= all_start[2]+min;
        i_4++;
        break;
        }
        if(i_4==3)
        {
        all_start[0]=vis_start[0];
        all_end[0]= all_start[0]+min;
        all_end[3]=vis_end[3];
        all_start[3]=all_end[3]-min;
        all_end[2]=Math.min((all_start[3]-recon),vis_end[2]);
        all_start[2]= all_end[2]-min;
        all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
        all_start[1]= all_end[1]-min;
        i_4=0;
        break;
        }
case 5:
        if(i_5==0)
        {
        all_start[0]=vis_start[0];
        all_end[0]= all_start[0]+min;
        all_end[4]=vis_end[4];
        all_start[4]=all_end[4]-min;
        all_start[1]=Math.max((all_end[0]+recon),
        vis_start[1]);
        all_end[1]= all_start[1]+min;
        all_start[2]=Math.max((all_end[1]+recon),
        vis_start[2]);
```

```
all_end[2]= all_start[2]+min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
i_5++;
break;
}
if(i_5==1)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[3]=Math.min((all_end[4]-recon),vis_end[3]);
all_start[3]= all_end[3]-min;
i_5++;
break;
}
if(i_5==2)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
i_5++;
break;
}
if(i_5==3)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[3]=Math.min((all_start[4]-recon),vis_end[3]);
all_start[3]= all_end[3]-min;
all_end[2]=Math.min((all_start[3]-recon),vis_end[2]);
all_start[2]= all_end[2]-min;
i_5++;
break;
}
```

```
if(i_5==4)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_end[1]=vis_end[1];
all_start[1]=vis_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
i_5++;
break;
}
if(i_5==5)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_end[1]=vis_end[1];
all_start[1]=vis_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[3]=Math.min((all_start[4]-recon),vis_end[3]);
all_start[3]= all_end[3]-min;
i_5++;
break;
}
if(i_5==6)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_start[3]=vis_start[3];
all_end[3]= all_start[3]+min;
all_end[2]=Math.min((all_start[3]-recon),vis_end[2]);
all_start[2]= all_end[2]-min;
all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
all_start[1]= all_end[1]-min;
i_5++;
break;
}
if(i_5==7)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_end[3]=Math.min((all_start[4]-recon),vis_end[3]);
all_start[3]= all_end[3]-min;
all_end[2] Math.min((all_start[3]-recon),vis_end[2]);
```

```
        all_start[2]= all_end[2]-min;
        all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
        all_start[1]= all_end[1]-min;
        i_5=0;
        break;
        }
case 6:
        if(i_6==0)
        {
        all_start[0]=vis_start[0];
        all_end[0]= all_start[0]+min;
        all_end[5]=vis_end[5];
        all_start[5]=all_end[5]-min;
        all_start[1]=Math.max((all_end[0]+recon),
        vis_start[1]);
        all_end[1]= all_start[1]+min;
        all_start[2]=Math.max((all_end[1]+recon),
        vis_start[2]);
        all_end[2]= all_start[2]+min;
        all_start[3]=Math.max((all_end[2]+recon),
        vis_start[3]);
        all_end[3]= all_start[3]+min;
        all_start[4]=Math.max((all_end[3]+recon),
        vis_start[4]);
        all_end[4]= all_start[4]+min;
        i_6++;
        break;
        }
        if(i_6==1)
        {
        all_start[0]=vis_start[0];
        all_end[0]= all_start[0]+min;
        all_end[5]= vis_end[5];
        all_start[5]=all_end[5]-min;
        all_start[1]=Math.max((all_end[0]+recon),
        vis_start[1]);
        all_end[1]= all_start[1]+min;
        all_start[2]=Math.max((all_end[1]+recon),
        vis_start[2]);
        all_end[2]= all_start[2]+min;
        all_start[3]=Math.max((all_end[2]+recon),
        vis_start[3]);
        all_end[3]= all_start[3]+min;
        all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
        all_start[4]= all_end[4]-min;
        i_6++;
        break;
        }
        if(i_6==2)
        {
        all_start[0]=vis_start[0];
        all_end[0]= all_start[0]+min;
        all_end[5]=vis_end[5];
        all_start[5]=all_end[5]-min;
        all_start[1]=Math.max((all_end[0]+recon),
        vis_start[1]);
        all_end[1]= all_start[1]+min;
```

```
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[3]=vis_end[3];
all_start[3]= all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
i_6++;
break;
}
if(i_6==3)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
all_end[3]=Math.min((all_start[4]-recon),vis_end[3]);
all_start[3]= all_end[3]-min;
i_6++;
break;
}
if(i_6==4)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[2]=vis_end[2];
all_start[2]= all_end[2]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
i_6++;
break;
}
if(i_6==5)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
all_start[1]=Math.max((all_end[0]+recon),
```

```
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[2]=vis_end[2];
all_start[2]= all_end[2]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
i_6++;
break;
}
if(i_6==6)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[2]=vis_end[2];
all_start[2]= all_end[2]-min;
all_end[3]=vis_end[3];
all_start[3]= all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
i_6++;
break;
}
if(i_6==7)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
all_end[3]=Math.min((all_start[4]-recon),vis_end[3]);
all_start[3]= all_end[3]-min;
all_end[2]=Math.min((all_start[3]-recon),vis_end[2]);
all_start[2]= all_end[2]-min;
i_6++;
break;
}
if(i_6==8)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
```

```
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
i_6++;
break;
}
if(i_6==9)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_end[4]=Math.min((all_start[5]-recon),
vis_end[4]);
all_start[4]= all_end[4]-min;
i_6++;
break;
}
if(i_6==10)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[3]=vis_end[3];
all_start[3]= all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
i_6++;
break;
}
if(i_6==11)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
```

```
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
all_end[3]=Math.min((all_start[4]-recon),vis_end[3]);
all_start[3]= all_end[3]-min;
i_6++;
break;
}
if(i_6==12)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_end[2]=vis_end[2];
all_start[2]= all_end[2]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
i_6++;
break;
}
if(i_6==13)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_end[2]=vis_end[2];
all_start[2]= all_end[2]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
i_6++;
break;
}
if(i_6==14)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
```

```
all_end[2]=vis_end[2];
all_start[2]= all_end[2]-min;
all_end[3]=vis_end[3];
all_start[3]= all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
i_6++;
break;
}
if(i_6==15)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[5]=vis_end[5];
all_start[5]=all_end[5]-min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
all_end[3]=Math.min((all_start[4]-recon),vis_end[3]);
all_start[3]= all_end[3]-min;
all_end[2]=Math.min((all_start[3]-recon),vis_end[2]);
all_start[2]= all_end[2]-min;
all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
all_start[1]= all_end[1]-min;
i_6=0;
break;
}
case 7:
if(i_7==0)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==1)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
```

```
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
all_start[5]= all_end[5]-min;
i_7++;
break;
}
if(i_7==2)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==3)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
```

```
all_start[5]= all_end[5]-min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
i_7++;
break;
}
if(i_7==4)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[3]=vis_end[3];
all_start[3]=all_end[3]-min;


all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==5)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[3]=vis_end[3];
all_start[3]=all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
all_start[5]= all_end[5]-min;
i_7++;
break;
}
if(i_7==6)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
```

```
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[3]=vis_end[3];
all_start[3]=all_end[3]-min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==7)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[5]=Math.min((all_start[6]-recon),
vis_end[5]);
all_start[5]= all_end[5]-min;
all_end[4]=Math.min((all_start[5]-recon),
vis_end[4]);
all_start[4]= all_end[4]-min;
all_end[3]=Math.min((all_start[4]-recon),
vis_end[3]);
all_start[3]= all_end[3]-min;
i_7++;
break;
}
if(i_7==8)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_start[4]=Math.max((all_end[3]+recon),
```

```
vis_start[4]);
all_end[4]= all_start[4]+min;
all_start[5]=Math.max((all_end[4]+recon),

vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==9)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_end[5]=Math.min((all_start[6]-recon),
vis_end[5]);
all_start[5]= all_end[5]-min;
i_7++;
break;
}
if(i_7==10)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==11)
{
all_start[0]=vis_start[0];
```

```
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
all_start[5]= all_end[5]-min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
i_7++;
break;
}
if(i_7==12)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[2]= vis_end[2];
all_start[2]=all_end[2]-min;
all_end[3]=vis_end[3];
all_start[3]=all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==13)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_end[3]=vis_end[3];
all_start[3]=all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
```

```
all_start[5]= all_end[5]-min;
i_7++;
break;
}
if(i_7==14)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_end[3]=vis_end[3];
all_start[3]= all_end[3]-min;
all_end[4]=vis_end[4];
all_start[4]= all_end[4]-min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==15)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_start[1]=Math.max((all_end[0]+recon),
vis_start[1]);
all_end[1]= all_start[1]+min;
all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
all_start[5]= all_end[5]-min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
all_end[3]=Math.min((all_start[4]-recon),vis_end[3]);
all_start[3]= all_end[3]-min;
all_end[2]=Math.min((all_start[3]-recon),vis_end[2]);
all_start[2]= all_end[2]-min;
i_7++;
break;
}
if(i_7==16)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_start[3]=Math.max((all_end[2]+recon),
```

```
vis_start[3]);
all_end[3]= all_start[3]+min;


all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==17)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
all_start[5]= all_end[5]-min;
i_7++;
break;
}
if(i_7==18)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
```

```
if(i_7==19)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
all_start[5]= all_end[5]-min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
i_7++;
break;
}
if(i_7==20)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[3]=vis_end[3];
all_start[3]=all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==21)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[3]=vis_end[3];
all_start[3]=all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
```

```
vis_start[4]);
all_end[4]= all_start[4]+min;

all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
all_start[5]= all_end[5]-min;
i_7++;
break;
}
if(i_7==22)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[3]=vis_end[3];
all_start[3]=all_end[3]-min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==23)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[1]=vis_end[1];
all_start[1]=all_end[1]-min;
all_start[2]=Math.max((all_end[1]+recon),
vis_start[2]);
all_end[2]= all_start[2]+min;
all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
all_start[5]= all_end[5]-min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
all_end[3]=Math.min((all_start[4]-recon),vis_end[3]);
all_start[3]= all_end[3]-min;
i_7++;
break;
}
if(i_7==24)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
```

```
all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
all_start[1]= all_end[1]-min;

all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==25)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
all_start[1]= all_end[1]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_end[5]=Math.min((all_start[6]-recon),
vis_end[5]);
all_start[5]= all_end[5]-min;
i_7++;
break;
}
if(i_7==26)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
all_start[1]= all_end[1]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_end[4]=vis_end[4];
all_start[4]=all_end[4]-min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
```

```
}
if(i_7==27)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
all_start[1]= all_end[1]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
all_start[5]= all_end[5]-min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
i_7++;
break;
}
if(i_7==28)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_end[1]=Math.min((all_start[2]-recon),
vis_end[1]);
all_start[1]= all_end[1]-min;
all_end[3]=vis_end[3];
all_start[3]=all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==29)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
all_start[1]= all_end[1]-min;
all_end[3]=vis_end[3];
all_start[3]=all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
```

```
}
if(i_7==27)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
all_start[1]= all_end[1]-min;
all_start[3]=Math.max((all_end[2]+recon),
vis_start[3]);
all_end[3]= all_start[3]+min;
all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
all_start[5]= all_end[5]-min;
all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
all_start[4]= all_end[4]-min;
i_7++;
break;
}
if(i_7==28)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_end[1]=Math.min((all_start[2]-recon),
vis_end[1]);
all_start[1]= all_end[1]-min;
all_end[3]=vis_end[3];
all_start[3]=all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
all_end[4]= all_start[4]+min;
all_start[5]=Math.max((all_end[4]+recon),
vis_start[5]);
all_end[5]= all_start[5]+min;
i_7++;
break;
}
if(i_7==29)
{
all_start[0]=vis_start[0];
all_end[0]= all_start[0]+min;
all_end[6]=vis_end[6];
all_start[6]=all_end[6]-min;
all_end[2]=vis_end[2];
all_start[2]=all_end[2]-min;
all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
all_start[1]= all_end[1]-min;
all_end[3]=vis_end[3];
all_start[3]=all_end[3]-min;
all_start[4]=Math.max((all_end[3]+recon),
vis_start[4]);
```

```
                    all_end[4]= all_start[4]+min;
                    all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
                    all_start[5]= all_end[5]-min;
                    i_7++;
                    break;
                    }
                    if(i_7==30)
                    {
                    all_start[0]=vis_start[0];
                    all_end[0]= all_start[0]+min;
                    all_end[6]=vis_end[6];
                    all_start[6]=all_end[6]-min;
                    all_end[2]=vis_end[2];
                    all_start[2]=all_end[2]-min;
                    all_end[1]=Math.min((all_start[2]-recon),vis_end[1]);
                    all_start[1]= all_end[1]-min;
                    all_end[3]=vis_end[3];
                    all_start[3]= all_end[3]-min;
                    all_end[4]=vis_end[4];
                    all_start[4]= all_end[4]-min;
                    all_start[5]=Math.max((all_end[4]+recon),
                    vis_start[5]);
                    all_end[5]= all_start[5]+min;
                    i_7++;
                    break;
                    }
                    if(i_7==31)
                    {
                    all_start[0]=vis_start[0];
                    all_end[0]= all_start[0]+min;
                    all_end[6]=vis_end[6];
                    all_start[6]=all_end[6]-min;
                    all_end[5]=Math.min((all_start[6]-recon),vis_end[5]);
                    all_start[5]= all_end[5]-min;
                    all_end[4]=Math.min((all_start[5]-recon),vis_end[4]);
                    all_start[4]= all_end[4]-min;
                    all_end[3]=Math.min((all_start[4]-recon),vis_end[3]);
                    all_start[3]= all_end[3]-min;
                    all_end[2]=Math.min((all_start[3]-recon),vis_end[2]);
                    all_start[2]= all_end[2]-min;
                    all_start[1]=Math.max((all_end[0]+recon),
                    vis_start[1]);
                    all_end[1]= all_start[1]+min;
                    i_7=0;
                    break;
                    }
                    vra[i] = new
Variable(vis_start,vis_end,all_start,all_end,sat_wt,sat_no);

                }

                varray =new Variable_array(vra);
                return(varray);
        }
}
```

6. InputClass.java

```
import java.io.*;

class InputClass
{
	int[] S,E,s_w,s_n;
	InputClass()
	{}
	InputClass(int[] vis_start,int[] vis_end,int[] sat_wt,int[]
sat_no)
	{
		S = new int[vis_start.length];
		E = new int[vis_end.length];
		s_w = new int[sat_wt.length];
		s_n = new int[sat_no.length];

		for(int i=0;i<S.length;i++)
		{
			S[i]=vis_start[i];
			E[i]=vis_end[i];
			s_w[i] = sat_wt[i];
			s_n[i] = sat_no[i];
		}

	}
}
```

7.MyTimer.java

```
import java.util.*;
class MyTimer extends GregorianCalendar
 {
	public long getTimeInMillis()
	{
		return super.getTimeInMillis();
	}

	public long difference(MyTimer t)
	{
		return this.getTimeInMillis() - t.getTimeInMillis();
	}
}
```

8.Neighborhood.java

```
class Variable_array
{
	Variable []vrba;
	Variable_array()
	{
```

```
	}
	Variable_array(Variable []vrb_a)
	{
		vrba = new Variable[vrb_a.length];
		vrba = vrb_a;
	}
}

class Neighborhood
{
	Neighborhood()
	{}
public static Variable_array neighborhood(Variable vb)
{

	Variable vrb = new Variable();
	Variable_array va;
	vrb=vb;
	Variable []vrba = new Variable[(vrb.S.length*4)];

	int step =10;

	for(int i=0;i<vrb.S.length;i++)
	{
		vrb.s[i]=vrb.s[i]+step;
		vrba[i]=new
Variable(vrb.S,vrb.E,vrb.s,vrb.e,vrb.s_w,vrb.s_n);
		vrb.s[i]=vrb.s[i]-step;
	}
	for(int i=0,j=vrb.S.length;i<vrb.S.length;i++,j++)
	{
		vrb.s[i]=vrb.s[i]-step;
		vrba[j]=new
Variable(vrb.S,vrb.E,vrb.s,vrb.e,vrb.s_w,vrb.s_n);
		vrb.s[i]=vrb.s[i]+step;
	}
	for(int i=0,j=vrb.S.length*2;i<vrb.S.length;i++,j++)
	{
		vrb.e[i]=vrb.e[i]+step;
		vrba[j]=new
Variable(vrb.S,vrb.E,vrb.s,vrb.e,vrb.s_w,vrb.s_n);
		vrb.e[i]=vrb.e[i]-step;
	}
	for(int i=0,j=vrb.S.length*3;i<vrb.S.length;i++,j++)
	{
		vrb.e[i]=vrb.e[i]-step;
		vrba[j]=new
Variable(vrb.S,vrb.E,vrb.s,vrb.e,vrb.s_w,vrb.s_n);
		vrb.e[i]=vrb.e[i]+step;
	}

	va = new Variable_array(vrba);

	return va;
}
}
```

APPENDIX 2
CODE FOR GREEDY SEARCH ALGORITHM

```
import java.lang.*;
import java.util.*;
import java.io.*;

class Greedy
{
public static void main(String args[])
throws Exception
{
MyTimer start = new MyTimer();
GregorianCalendar gcal = new GregorianCalendar();
System.out.print("Time Before: ");
System.out.print(gcal.get(Calendar.HOUR)+" :");
System.out.print(gcal.get(Calendar.MINUTE)+" :");
System.out.println(gcal.get(Calendar.SECOND));

double optimum=0;
int length_of_fine_tuning =5;

int[] vis_start = new int[4];
int[] vis_end = new int[4];
int[] sat_wt = new int[4];
int[] sat_no = new int[4];

vis_start[0]=6520;
vis_start[1]=7244;
vis_start[2]=7712;
vis_start[3]=8027;

vis_end[0]=7115;
vis_end[1]=8318;
vis_end[2]=8650;
vis_end[3]=9196;

sat_wt[0]=1;
sat_wt[1]=4;
sat_wt[2]=3;
sat_wt[3]=2;

sat_no[0]=1;
sat_no[1]=2;
sat_no[2]=3;
sat_no[3]=4;

Vector v;
Vector v1;
Vector v2;

Variable_array varray = new Variable_array();
InputClass iclass = new InputClass(vis_start,vis_end,sat_wt,sat_no);
InitialSolution1Modified insol = new InitialSolution1Modified();
//InitialSolution1 insol = new InitialSolution1();

FileWriter fw = new FileWriter("clash_solutions");
FineTuning ft = new FineTuning();
Evaluation evl = new Evaluation();
```

```
varray = insol.initialsolution1modified(iclass);
//varray = insol.initialsolution1(iclass);

Variable vrb_optimum= new Variable();
Variable_array input;
Variable []input_a;
double []input_d;
Vector v3=new Vector();
Vector v4=new Vector();

//varray.vrba.length

System.out.println("length ="+varray.vrba.length);

for(int m=1;m<varray.vrba.length;m++)
{
double finalmax=0,previous=0,next=0,
index_previous=0,index_next=0;
boolean flag=true,flag1,flag2=true;
double answer=0,answer1;
double []temp;
int iter=10000,prob,count1=0;

Random rm = new Random((long)10.00);
Variable vrb= new Variable();
Variable vrb_final = new Variable();
Feasibility fb = new Feasibility();
Evaluation ev = new Evaluation();
Neighborhood nh = new Neighborhood();
Variable_array  vr_array = new Variable_array();

vrb=varray.vrba[m];

for(int j=0;j<iter;j++)
{
      vr_array = nh.neighborhood(vrb);
      v = new Vector();
      v1 = new Vector();
      flag = fb.feasibility(vrb);
      if(flag)
      {
      answer=ev.evaluation(vrb);
      //System.out.println("answer="+answer);
      fw.write(""+answer+"\n");
      if(answer>finalmax)
      {
      finalmax=answer;
      vrb_final=vrb;
      }
      previous=next;
      next = finalmax;
      if(previous==next)
      {
      if(flag2)
      {
      flag2=false;
      index_previous = j;
      index_next=j;
```

```
        }
        if((count1==10)&&((index_next-index_previous)==9))
        break;
        else
        {
        if(count1==10)
        {
        flag2=true;
        count1=0;
        }
        }
//System.out.println("Now let's check the neighborhood solutions");
        for(int i=0;i<vr_array.vrba.length;i++)
        {
        flag1 = fb.feasibility(vr_array.vrba[i]);
//System.out.println("Is solution feasible ? (true/false) :->
"+flag1);
        if(flag1)
        {
        answer1=ev.evaluation(vr_array.vrba[i]);
        //System.out.println("answer1="+answer1);
        v.addElement(new Double(answer1-answer));
        v1.addElement(new Integer(i));
        }
        }
        Enumeration vEnum = v.elements();
        double max=-12000;
        int element=0,index=0,index1=0;
        v2 = new Vector();
        temp = new double[v.size()];
        while(vEnum.hasMoreElements())
        {
        //System.out.println("vEnum:= "+vEnum.nextElement());
temp[element] = Double.parseDouble(vEnum.nextElement().toString());
        if(temp[element]>=max)
        {
        if(temp[element]>max)
        {
        max=temp[element];
        index = element;
        }
        else
        {
        max=temp[element];
        v2.addElement(new Integer(element));
        }
        }
        element++;
        }
        //System.out.println("max = "+max);
        if(v1.isEmpty()==false)
        {
        if(v2.isEmpty()==true)
        vrb = vr_array.vrba[Integer.parseInt
        (v1.elementAt(index).toString())];
        else
        {
        //System.out.println("size:=  "+v2.size());
```

```
//System.out.println("interm.. "+interm);
prob=(int)interm;
//System.out.println("prob:= " +prob);
if(prob==v2.size())
index1=index;
else
index1=Integer.parseInt(v2.elementAt(prob).toString());
vrb = vr_array.vrba[Integer.parseInt
(v1.elementAt(index1).toString())];
}
}
else
break;
}
else
break;
}
if(flag)
{
for(int i=0;i<vrb_final.s.length;i++)
v3.addElement(vrb_final);
v4.addElement(new Double(finalmax));
}
else
{}
}

input_a = new Variable[v3.size()];
input_d = new double[v4.size()];
for(int i=0;i<input_d.length;i++)
{
input_d[i] =Double.parseDouble(v4.elementAt(i).toString());
input_a[i] =(Variable)v3.elementAt(i);
}
double temp1;
Variable temp2 = new Variable();
for(int i=0;i<input_d.length;i++)
{
for(int j=i;j<input_d.length;j++)
{
if(input_d[j]>input_d[i])
{
temp1 = input_d[j];
input_d[j]= input_d[i];
input_d[i]=temp1;
temp2 = input_a[j];
input_a[j]= input_a[i];
input_a[i]=temp2;
}
}
}
Variable []input_a1 = new Variable[length_of_fin

for(int i=0;i<length_of_fine_tuning;i++)
input_a1[i] = input_a[i];
input = new Variable_array(input_a1);
vrb_optimum = ft.fineTuning(input);
optimum = ev1.evaluation(vrb_optimum);
```

```
        for(int opt=0;opt<vrb_optimum.s.length;opt++)
        System.out.println(vrb_optimum.s[opt]+"\t"+vrb_optimum.e[opt]+"
\t"+vrb_optimum.s_n[opt]);
        System.out.println("optimum:= "+optimum);

        fw.close();

        GregorianCalendar gcal1 = new GregorianCalendar();
        System.out.print("Time After: ");
        System.out.print(gcal1.get(Calendar.HOUR)+" :");
        System.out.print(gcal1.get(Calendar.MINUTE)+" :");
        System.out.println(gcal1.get(Calendar.SECOND));

        MyTimer stop = new MyTimer();
        long millis = stop.difference(start);
        System.out.println("Program Took: " + millis + "
milliseconds");
        }
}
```

APPENDIX 3
CODES FOR TABU SEARCH ALGORITHM

1.Tabu.java

```
import java.lang.*;
import java.util.*;
import java.io.*;

class Tabu
{
public static void main(String args[])
throws Exception
{
MyTimer start = new MyTimer();
GregorianCalendar gcal = new GregorianCalendar();

System.out.print("Time Before: ");
System.out.print(gcal.get(Calendar.HOUR)+" :");
System.out.print(gcal.get(Calendar.MINUTE)+" :");
System.out.println(gcal.get(Calendar.SECOND));

double optimum=0;
int[] vis_start = new int[4];
int[] vis_end = new int[4];
int[] sat_wt = new int[4];
int[] sat_no = new int[4];

vis_start[0]=6520;
vis_start[1]=7244;
vis_start[2]=7712;
vis_start[3]=8027;

vis_end[0]=7115;
vis_end[1]=8318;
vis_end[2]=8650;
vis_end[3]=9196;

sat_wt[0]=1;
sat_wt[1]=2;
sat_wt[2]=3;
sat_wt[3]=4;

sat_no[0]=1;
sat_no[1]=2;
sat_no[2]=3;
sat_no[3]=4;

Vector v;
Vector v1;
Vector v2;

Variable_array varray = new Variable_array();
InputClass iclass = new InputClass(vis_start,vis_end,sat_wt,sat_no);

//InitialSolutionModified insol = new InitialSolutionModified();
InitialSolution insol = new InitialSolution();

FileWriter fw = new FileWriter("clash_solutions");
FineTuning ft = new FineTuning();
Evaluation evl = new Evaluation();
```

```
//varray = insol.initialsolutionmodified(iclass);
varray = insol.initialsolution(iclass);
//System.out.println("I am here");

Variable vrb_optimum = new Variable();
Variable_array input;
Variable []input_a;
double []input_d;
Vector v3=new Vector();
Vector v4=new Vector();
int finetuning_array_size =3;
int tabu_tenure = 1;

//varray.vrba.length
for(int m=1;m<varray.vrba.length;m++)
{
double finalmax=0,previous=0,next=0,index_previous=0,index_next=0;
boolean flag,flag1,flag2=true;
double answer=0,answer1;
double []temp;
int iter=150,count1=0;

Variable vrb= new Variable();
Variable vrb_final = new Variable();
Feasibility fb = new Feasibility();
Evaluation ev = new Evaluation();
Neighborhood nh = new Neighborhood();
Variable_array  vr_array = new Variable_array();
Variable_array  vr_array_feasible = new Variable_array();
Select sl = new Select();
TabuList tl = new TabuList();
Aspiration ap = new Aspiration();
boolean terminate = false;
vrb=varray.vrba[m];

flag = fb.feasibility(vrb);
//System.out.println(flag);
if(flag)
{
for(int j=0;j<iter;j++)
{
//System.out.println(" iter:= " +j);
answer=ev.evaluation(vrb);
/*System.out.println(" answer:= " +answer);
System.out.println(" finalmax:= " +finalmax);*/
if(answer>finalmax)
{
      finalmax=answer;
      vrb_final=vrb;
}
previous=next;
next = finalmax;
if(previous==next)
{
if(flag2)
{
flag2=false;
index_previous = j;
index_next=j;
}
count1++;
```

```
index_next=j;
}
if((count1==20)&&((index_next-index_previous)==19))
break;
else
{
if(count1==20)
{
flag2=true;
count1=0;
}
}
vr_array = nh.neighborhood(vrb);
vr_array_feasible = sl.select(vrb,vr_array);

//System.out.println("length=="+vr_array_feasible.vrba.length);
if(vr_array_feasible.vrba.length==0)
break;
else
{
for(int i=0;i<vr_array_feasible.vrba.length;i++)
{
boolean tabu_condition = tl.isTabu(vrb,vr_array_feasible.vrba[i]);
//System.out.println("tabu_condition = "+tabu_condition);
if(tabu_condition)
{boolean aspiration_condition =
ap.isAspirant(vr_array_feasible.vrba[i]);
//System.out.println("aspiration_condition = "+aspiration_condition);
if(aspiration_condition)
{
String tabu_element= tl.comparision(vrb,vr_array_feasible.vrba[i]);
tl.addToTabuList(tabu_element);
vrb = vr_array_feasible.vrba[i];
break;
}
}
else
{
String tabu_element= tl.comparision(vrb,vr_array_feasible.vrba[i]);
//System.out.println("tabu_element = "+tabu_element);

tl.addToTabuList(tabu_element);
vrb = vr_array_feasible.vrba[i];
break;
}
if(i==(vr_array_feasible.vrba.length-1))
terminate = true;
}
if(j>=tabu_tenure)
tl.removeFromTabuList();
if(terminate)
break;
}
}
v3.addElement(vrb_final);
v4.addElement(new Double(finalmax));
}
else
{}
```

```
}
System.out.println("size=="+v3.size());
input_a = new Variable[v3.size()];
input_d = new double[v4.size()];

//System.out.println("well,I am here");
for(int i=0;i<input_d.length;i++)
{
input_d[i] =Double.parseDouble(v4.elementAt(i).toString());
input_a[i] =(Variable)v3.elementAt(i);
}
double temp1;
Variable temp2 = new Variable();
for(int i=0;i<input_d.length;i++)
{
for(int j=i;j<input_d.length;j++)
{
if(input_d[j]>input_d[i])
{
temp1 = input_d[j];
input_d[j]= input_d[i];
input_d[i]=temp1;
temp2 = input_a[j];
input_a[j]= input_a[i];
input_a[i]=temp2;
}
}
}

Variable []input_a1 = new Variable[finetuning_array_size];
for(int i=0;i<finetuning_array_size;i++)
input_a1[i] = input_a[i];
input = new Variable_array(input_a1);
vrb_optimum = ft.fineTuning(input);
optimum = ev1.evaluation(vrb_optimum);
for(int opt=0;opt<vrb_optimum.s.length;opt++)
System.out.println(vrb_optimum.s[opt]+"\t"+vrb_optimum.e[opt]+"\t"+vr
b_optimum.s_n[opt]);

System.out.println("optimum:= "+optimum);

GregorianCalendar gcal1 = new GregorianCalendar();

System.out.print("Time After: ");
System.out.print(gcal1.get(Calendar.HOUR)+" :");
System.out.print(gcal1.get(Calendar.MINUTE)+" :");
System.out.println(gcal1.get(Calendar.SECOND));

MyTimer stop = new MyTimer();
long millis = stop.difference(start);
System.out.println("Program Took: " + millis + " milliseconds");

}
}
```

2. Aspiration.java

```
class Aspiration
{
	static double previous_best;
	Aspiration()
	{
		previous_best = 0;
	}
	public static boolean isAspirant(Variable vr)
	throws Exception
	{

		Variable vrb = new Variable();
		vrb=vr;
		Evaluation ev = new Evaluation();

		double this_solution = ev.evaluation(vrb);

		/*System.out.println("this_sol:="+this_solution);
		System.out.println("pre_best:="+previous_best);*/


		if(this_solution>previous_best)
		{
			previous_best=this_solution;
			return(true);
		}
		else
			return(false);
	}


}
```

3.Select.java

```
import java.util.*;
import java.io.*;

class Select
{
Select()
{}

public static Variable_array select(Variable vr,Variable_array vrr)
throws Exception
{
	Variable_array varray = new Variable_array();
	Variable_array varray_return;

	Variable vrb = new Variable();
	Feasibility fb = new Feasibility();
	vrb = vr;
	varray = vrr;

	Variable []vrb_temp;
	double []feasible_array;
	Evaluation ev = new Evaluation();
	double answer = ev.evaluation(vrb);
```

```
        Vector v = new Vector();
        Vector v1 = new Vector();

        for(int i=0;i<varray.vrba.length;i++)
        {
        boolean flag1 = fb.feasibility(varray.vrba[i]);
        if(flag1)
        {
        double answer1=ev.evaluation(varray.vrba[i]);
        v.addElement(new Double(answer1-answer));
        v1.addElement(varray.vrba[i]);
        }
        }
        Enumeration vEnum = v.elements();
        Enumeration v1Enum = v1.elements();

        feasible_array = new double[v.size()];
        vrb_temp = new Variable[v1.size()];

        int element = 0;

        while(vEnum.hasMoreElements())
        {
        feasible_array[element] =
        Double.parseDouble(vEnum.nextElement().toString());
        vrb_temp[element]=(Variable)v1Enum.nextElement();
        element++;
        }

        int u=0;
        double temp1;
        Variable temp2 =new Variable();

        for(int i=0;i<feasible_array.length-1;i++)
        {
        for(int j=i+1;j<feasible_array.length;j++)
        {
        if(feasible_array[j]>feasible_array[i])
        {
        temp1 = feasible_array[j];
        feasible_array[j]= feasible_array[i];
        feasible_array[i]=temp1;
        temp2 = vrb_temp[j];
        vrb_temp[j]= vrb_temp[i];
        vrb_temp[i]=temp2;
        }
        }
        }

        varray_return = new Variable_array(vrb_temp);
        return(varray_return);
        }
}
```

4.TabuList.java

```
import java.io.*;
import java.lang.*;
import java.util.*;
```

```
class TabuList
{
TabuList()
{}
static Vector v  =  new  Vector();

public static boolean isTabu(Variable cur_sol,Variable new_sol)
{
      Variable current_solution = new Variable();
      Variable new_solution = new Variable();
      boolean ret;
      current_solution = cur_sol;
      new_solution = new_sol;
      String index = comparision(current_solution, new_solution);
      if(v.contains(index)==true)
      {ret = true;}
      else
      {ret = false;}
      return(ret);
}
public static String comparision(Variable cur_sol, Variable new_sol)
{
      Variable current_solution = new Variable();
      Variable new_solution = new Variable();
      current_solution = cur_sol;
      new_solution = new_sol;
      String sr =" ";
      for(int i=0;i<current_solution.e.length;i++)
      {
      if(current_solution.s[i]!=new_solution.s[i])
      {
      sr="s"+i;
      break;
      }
      if(current_solution.e[i]!=new_solution.e[i])
      {
      sr="e"+i;
      break;
      }
      }
      return (sr);
}
public static void addToTabuList(String s)
{
      String str = s;
      v.addElement(str);
}

public static void removeFromTabuList()
{
      String removed =(String) v.remove(0);
      //System.out.println("yes removed");
}
}
```

APPENDIX 4
CODE FOR SIMULATED ANNEALING

1. Simulated Annealing.java

```
import java.lang.*;
import java.util.*;
import java.io.*;

class SA
{
public static void main(String args[])
throws Exception
{
      MyTimer start = new MyTimer();
      GregorianCalendar gcal = new GregorianCalendar();
      System.out.print("Time Before: ");
      System.out.print(gcal.get(Calendar.HOUR)+" :");
      System.out.print(gcal.get(Calendar.MINUTE)+" :");
      System.out.println(gcal.get(Calendar.SECOND));

      int[] vis_start = new int[4];
      int[] vis_end = new int[4];
      int[] sat_wt = new int[4];
      int[] sat_no = new int[4];

      vis_start[0]=6520;
      vis_start[1]=7244;
      vis_start[2]=7712;
      vis_start[3]=8027;

      vis_end[0]=7115;
      vis_end[1]=8318;
      vis_end[2]=8650;
      vis_end[3]=9196;

      sat_wt[0]=1;
      sat_wt[1]=3;
      sat_wt[2]=4;
      sat_wt[3]=2;

      sat_no[0]=1;
      sat_no[1]=2;
      sat_no[2]=3;
      sat_no[3]=4;

Variable_array varray = new Variable_array();
InputClass iclass = new InputClass(vis_start,vis_end,sat_wt,sat_no);
//InitialSolutionModified insol = new InitialSolutionModified();
InitialSolution insol = new InitialSolution();
FineTuning ft = new FineTuning();
Evaluation ev1 = new Evaluation();
//varray = insol.initialsolutionmodified(iclass);
varray = insol.initialsolution(iclass);
Variable vrb_optimum= new Variable();
Variable vrb_final = new Variable();
Variable vrb_initial = new Variable();
Variable vrb = new Variable();
Evaluation ev = new Evaluation();
```

```
Variable_array input;
double T,epsilon,r;
int iter=0,iter1=0;
double solution=0,solution_optimum=0;
int fine_tune_length = 5;

Variable []input_a;
double []input_d;
Vector v3=new Vector();
Vector v4=new Vector();
FileWriter fw = new FileWriter("clash_solutions");

for(int m=0;m<varray.vrba.length;m++)
{
boolean flag=true;
T=25;
epsilon =1;
r=.99;

long seed1 = 7;
long seed2 = 10;

vrb=varray.vrba[m];
vrb_initial=varray.vrba[m];
Feasibility fb = new Feasibility();
Neighborhood nh = new Neighborhood();
Variable_array  vr_array = new Variable_array();
Variable_array  vr_array_feasible;
Select slt = new Select();

Random rm = new Random(seed1);
Random rm_new = new Random(seed2);

while(T>=epsilon)
{
iter=0;
while(iter<50)
{
iter++;
vr_array = nh.neighborhood(vrb);
flag = fb.feasibility(vrb);
//System.out.println("Is solution feasible ? (true/false) :->
"+flag);
if(flag)
{
int feasibility_count=0;
for(int f=0;f<vr_array.vrba.length;f++)
{
if(fb.feasibility(vr_array.vrba[f]))
feasibility_count++;
}
Variable []vrb_temp = new Variable[feasibility_count];
int f_count = 0;
for(int f=0;f<vr_array.vrba.length;f++)
{
if(fb.feasibility(vr_array.vrba[f]))
{
vrb_temp[f_count]= vr_array.vrba[f];
f_count++;
}
}
```

```
vr_array_feasible = new Variable_array(vrb_temp);
vrb = slt.select(vrb,vr_array_feasible,T,rm.nextDouble(),
rm_new.nextDouble());
double tempsol = ev.evaluation(vrb);
if(solution<=tempsol)
{
solution =tempsol;
vrb_final = vrb;
}
}
else
break;
}
fw.write(""+solution+"\n");
T=T*r;
iter1++;
}

v3.addElement(vrb_final);
v4.addElement(new Double(solution));
}

input_a = new Variable[v3.size()];
input_d = new double[v4.size()];
for(int i=0;i<input_d.length;i++)
{
input_d[i] =Double.parseDouble(v4.elementAt(i).toString());
input_a[i] =(Variable)v3.elementAt(i);
}
double temp1;
Variable temp2 = new Variable();

for(int i=0;i<input_d.length;i++)
{
for(int j=i;j<input_d.length;j++)
{
if(input_d[j]>input_d[i])
{
      temp1 = input_d[j];
      input_d[j]= input_d[i];
      input_d[i]=temp1;
      temp2 = input_a[j];
      input_a[j]= input_a[i];
      input_a[i]=temp2;
}
}
}
Variable []input_a1 = new Variable[fine_tune_length];
for(int i=0;i<fine_tune_length;i++)
input_a1[i] = input_a[i];

input = new Variable_array(input_a1);

vrb_optimum = ft.fineTuning(input);
solution_optimum = ev1.evaluation(vrb_optimum);
for(int i=0;i<vrb_optimum.s.length;i++)
{
System.out.println(vrb_optimum.s[i]+"\t"+vrb_optimum.e[i]+"\t"+vrb_op
timum.s_n[i]);
}
```

```
System.out.println("sol_opt:="+solution_optimum);

GregorianCalendar gcal1 = new GregorianCalendar();

System.out.print("Time After: ");
System.out.print(gcal1.get(Calendar.HOUR)+" :");
System.out.print(gcal1.get(Calendar.MINUTE)+" :");
System.out.println(gcal1.get(Calendar.SECOND));

MyTimer stop = new MyTimer();
long millis = stop.difference(start);
System.out.println("Program Took: " + millis + " milliseconds");
}
}
```

2.Select.java

```
import java.util.*;
import java.io.*;

class Select
{
Select()
{}
public static Variable select(Variable vr,Variable_array vrr,double
t,double rm1,double rm2)
throws Exception
{

Variable_array varray = new Variable_array();
Variable vrb = new Variable();
Variable vrb_temp = new Variable();
double T =t,prob1,random1=rm1,random2=rm2;
int prob;

vrb = vr;
varray = vrr;

Evaluation ev = new Evaluation();
double interm =((random1)*(varray.vrba.length));
prob=(int)interm;
if(prob==varray.vrba.length)
vrb_temp=varray.vrba[prob-1];
else
vrb_temp=varray.vrba[prob];
double difference;
difference = (ev.evaluation(vrb_temp))-(ev.evaluation(vrb));

if(difference >= 0)
      return(vrb_temp);
else
{
      double exponent = difference/t;
      double exponent_value = Math.pow(2.718281,exponent);
      if(random2<=exponent_value)
      return(vrb_temp);
      else
      return(vrb);
}
}
}
```

APPENDIX 5
CODE FOR RESOLVING CLASHES IN THE ENTIRE VISIBILITY FILE WITH ONE OF THE ABOVE ALGORITHM

ClashResolve.java

```
import java.lang.*;
import java.io.*;
import java.util.*;

//THIS PROGRAM PICKS UP THE CLASH SITUATION FROM THE
// FILE CLASDID1.TXT AND SENDS IT FOR CLASH RESOLUTION
//AFTER RESOLVING CLASHES, IT REPLACES THE VISIBILITY START
//AND VISIBILITY END TIMINGS WITH ALLOCATION START AND
//ALLOCATION END TIMINGS.

class TestArray11
{
String []s = new String[19];
TestArray11()
{}

TestArray11(String[] str)
{
for(int i=0;i<19;i++)
s[i]=str[i];
}
}

class ClashResolve
{
public static void main(String args[])
throws Exception
{
      MyTimer start = new MyTimer();
      GregorianCalendar gcal = new GregorianCalendar();
      System.out.print("Time Before: ");
      System.out.print(gcal.get(Calendar.HOUR)+" :");
      System.out.print(gcal.get(Calendar.MINUTE)+" :");
      System.out.println(gcal.get(Calendar.SECOND));

      String sv1,sv2,name;
      StringTokenizer stk,stk1,stk2;
      TestArray11 oav1;

      int no_of_ttc_stations = 12;
      String [] station_name = new String[no_of_ttc_stations];

      File f1 = new File("clashID1");
      File f_ttc = new File("ttcstation.txt");

      FileReader fr1 = new FileReader(f1);
      FileReader fr_ttc = new FileReader(f_ttc);
      BufferedReader br1 = new BufferedReader(fr1);
      BufferedReader br_ttc = new BufferedReader(fr_ttc);
      FileWriter fw1 = new FileWriter("clashID2");

      Vector v = new Vector();
      while((sv1=br1.readLine())!=null)
      {
```

```
stk1 = new StringTokenizer(sv1);
String []sav1 = new String[stk1.countTokens()];

int tnv=0;
while(stk1.hasMoreTokens())
{
sav1[tnv]=stk1.nextToken();
tnv++;
}

oav1 = new TestArray11(sav1);
v.addElement(oav1);
}
v.trimToSize();
int count =0;
while((name=br_ttc.readLine())!=null)
{
station_name[count]= name;
count++;
}

for(int k=0;k<(v.size());k++)
{
boolean isttc = false;
for(int i=0;i<no_of_ttc_stations;i++)
{
if((station_name[i]).equals(((TestArray11)v.elementAt(k)).s[5].
toString()))
{
      isttc = true;
      //System.out.println("if condition is satisfied");
      break;
}
}
//System.out.println("I am here"+isttc+k);
if(isttc)
{
if((((TestArray11)v.elementAt(k)).s[15]).equals("FREE"))
{
Vector v1 = new Vector();
v1.addElement(v.elementAt(k));
for(int i=k+1;i<k+100;i++)
{
if(i<v.size())
{
if(((TestArray11)v.elementAt(i)).s[5].equals(((TestArray11)v.el
ementAt(k)).s[5]))
{
if(((TestArray11)v.elementAt(i)).s[0].equals(((TestArray11)v.el
ementAt(k)).s[0]))
{
v1.addElement(v.elementAt(i));
}
else
break;
}
if(v1.size()>8)
break;
}
else
break;
```

```
        }
        if(v1.size()>1)
        {
        int[] vis_start = new int[v1.size()];
        int[] vis_end = new int[v1.size()];
        int[] s_w = new int[v1.size()];
        int[] s_n = new int[v1.size()];
        long temp_date =0;
        for(int i=0;i<v1.size();i++)
        {
        int  date = Integer.parseInt(((TestArray11)
        v1.elementAt(i)).s[3].toString());
        int month = Integer.parseInt(((TestArray11)
        v1.elementAt(i)).s[2].toString());
        int year = Integer.parseInt(((TestArray11)
        v1.elementAt(i)).s[1].toString());
        int vis_start_hr =Integer.parseInt(((TestArray11)
        v1.elementAt(i)).s[8].toString());
        int vis_start_mt = Integer.parseInt(((TestArray11)
        v1.elementAt(i)).s[9].toString());
        int vis_start_sec = Integer.parseInt(((TestArray11)
        v1.elementAt(i)).s[10].toString());
        int vis_end_hr =Integer.parseInt(((TestArray11)
        v1.elementAt(i)).s[11].toString());
        int vis_end_mt = Integer.parseInt(((TestArray11)
        v1.elementAt(i)).s[12].toString());
        int vis_end_sec = Integer.parseInt(((TestArray11)
        v1.elementAt(i)).s[13].toString());
        Calendar cal = Calendar.getInstance();
        Calendar cal1 = Calendar.getInstance();

        cal.set(year,month,date,vis_start_hr,vis_start_mt,vis_start_sec
);

        cal1.set(year,month,date,vis_end_hr,vis_end_mt,vis_end_sec);


        if((vis_start_hr==23)&&(vis_end_hr==0))
        cal1.add(Calendar.DATE,1);
        Date before = cal.getTime();
        long bef = before.getTime();
        if(i==0)
        temp_date = bef;
        Date after = cal1.getTime();
        long aft = after.getTime();
        //System.out.println("bef is: "+bef);
        //System.out.println(" aft is:"+aft);
        vis_start[i] =(int)((bef-temp_date)/1000);
        vis_end[i] = (int)((aft-temp_date)/1000);
        s_n[i] = Integer.parseInt(((TestArray11)
        v1.elementAt(i)).s[18]);
        s_w[i] = Integer.parseInt(((TestArray11)
        v1.elementAt(i)).s[17]);
        }
        //pass this objet to clash resolution and get an variale object
back..//get them converted into
        //date and time format.
        //compared variable and input class objects...

        InputClass iclass = new InputClass(vis_start,vis_end,s_w,s_n);
        //one can the below line to tabu or simulated annealing.
```

```
Greedy1 greedy1 = new Greedy1();
Variable vrb = greedy1.greedy1(iclass);

for(int i=0;i<iclass.s_n.length;i++)
{
boolean match = false;
for(int j=0;j<vrb.s.length;j++)
{
if((iclass.s_n[i]) == (vrb.s_n[j]))
{
match=true;
break;
}
}
if(match==false)
{
String []temp = new String[19];
for(int j=0;j<15;j++)
temp[j] = (((TestArray11)v.elementAt(iclass.s_n[i])).s[j]);
temp[15] = "Clash-Resolved/NS";
for(int j=16;j<19;j++)
temp[j] = (((TestArray11)v.elementAt(iclass.s_n[i])).s[j]);
TestArray11 ts1 = new TestArray11(temp);
v.setElementAt(ts1,iclass.s_n[i]);
}
}
int []new_start_hour = new int[vrb.s.length];
int []new_start_minute = new int[vrb.s.length];
int []new_start_second = new int[vrb.s.length];
int []new_end_hour = new int[vrb.s.length];
int []new_end_minute = new int[vrb.s.length];
int []new_end_second = new int[vrb.s.length];

for(int i=0;i<vrb.s.length;i++)
{
Calendar cal_new = Calendar.getInstance();
Date dt = new Date((long)((vrb.s[i]*1000)+temp_date));
cal_new.setTime(dt);
new_start_hour[i] = cal_new.get(Calendar.HOUR_OF_DAY);
new_start_minute[i] = cal_new.get(Calendar.MINUTE);
new_start_second[i] = cal_new.get(Calendar.SECOND);

Calendar cal_new1 = Calendar.getInstance();
Date dt1 = new Date((long)((vrb.e[i]*1000)+temp_date));
cal_new1.setTime(dt1);
new_end_hour[i] = cal_new1.get(Calendar.HOUR_OF_DAY);
new_end_minute[i] = cal_new1.get(Calendar.MINUTE);
new_end_second[i] = cal_new1.get(Calendar.SECOND);
}
for(int i=0;i<vrb.s.length;i++)
{
String []temp = new String[19];
for(int j=0;j<8;j++)
temp[j] = ((TestArray11)
v.elementAt(vrb.s_n[i])).s[j].toString();
temp[8] =  (""+new_start_hour[i]);
temp[9] = (""+new_start_minute[i]);
temp[10] = (""+new_start_second[i]);
temp[11] = (""+new_end_hour[i]);
temp[12] = (""+new_end_minute[i]);
temp[13] = (""+new_end_second[i]);
```

```
        temp[14] = ((TestArray11)
        v.elementAt(vrb.s_n[i])).s[14].toString();
        temp[15] = "CR_TTC";
        for(int j=16;j<19;j++)
        temp[j] = ((TestArray11)
        v.elementAt(vrb.s_n[i])).s[j].toString();
        TestArray11 ts1 = new TestArray11(temp);
        v.setElementAt(ts1,vrb.s_n[i]);
        }
        }//if v.size end
        }//if free end
        }//if isttc end
        }//for loop end

        for(int i=0;i<v.size();i++)
        {
              for(int j=0;j<19;j++)
                    fw1.write(((TestArray11)v.elementAt(i)).s[j]+"\t");
              fw1.write("\n");
        }


              GregorianCalendar gcal1 = new GregorianCalendar();

              System.out.print("Time After: ");
              System.out.print(gcal1.get(Calendar.HOUR)+" :");
              System.out.print(gcal1.get(Calendar.MINUTE)+" :");
              System.out.println(gcal1.get(Calendar.SECOND));

              MyTimer stop = new MyTimer();
              long millis = stop.difference(start);
System.out.println("Program Took: " + millis + " milliseconds");

fw1.close();
fr1.close();

}
}
```